ACM ISBN: 978-1-4503-0672-0

Twenty-Seventh Annual COMPUTER SECURITY APPLICATIONS Conference

Twenty-Seventh Annual

COMPUTER SECURITY APPLICATIONS Conference

Orlando, Florida
5-9 December 2011

Sponsored by Applied Computer Security Associates

Proceedings

27th Annual Computer Security Applications Conference

ACSAC 2011

Proceedings

27th Annual Computer Security Applications Conference

Orlando, Florida, USA
5–9 December 2011

Sponsored by
Applied Computer Security Associates

The Association for Computing Machinery
2 Penn Plaza, Suite 701
New York, New York 10121-0701

ACM ISBN: 978-1-4503-0672-0

Editorial production by Christoph Schuba
Published by ACM, Inc. within the ACM International Conference Proceedings Series

Table of Contents

Message from the Conference Chair

Welcome to the 27th Annual Computer Security Applications Conference (ACSAC). Since last held in Orlando nearly two decades ago, ACSAC continues to be recognized as one of the leading security events in the world, and is one of the longest-running events in the field. Our unique blend of learning and networking, with equal participation from industry, government, and academia, has served as a forum for groundbreaking new work as well as fostered discussion on immediately applicable security techniques.

Every year, ACSAC receives hundreds of submissions from around the world. The technical papers are peer-reviewed by leading experts in the security field in a double-blind process to ensure fairness, and the very best are accepted for presentation and inclusion in the proceedings. Leading this effort for 2011 is program chair John McDermott along with co-chair Michael Locasto, both of whom have done an outstanding job to ensure a quality program across a diverse range of security topics – from social network security and applied cryptography to situational awareness and mobile security.

In addition to the technical paper presentations, ACSAC invites select guest speakers to provide stimulating presentations on topics of wide interest. This year's Distinguished Practitioner is Susan Landau from Harvard University, and the Invited Essayist is Terry Benzel from the USC Information Sciences Institute. Providing a look back at past groundbreaking work and how the security field has evolved since, are Classic Paper presenters Paul Syverson from the US Naval Research Laboratory, and Matt Blaze from the University of Pennsylvania. Thank you to Jeremy Epstein for coordinating the selection of our guest speakers.

Rounding out the technical program are case studies, panels, works-in-progress, and posters – each of these present additional opportunities for attendees to learn from and engage security practitioners and researchers. Perennial panel favorites, such as the *New Security Paradigms Workshop Experience*, are being complemented by new panels including *The Search for Meaningful Trustworthiness* and *Software Assurance in the Globalised Era*. Case studies are once again providing industry application examples of security and best practices, ranging from smart phone security architecture to Internet backbone worm detection. Reverting back to their prior format, works-in-progress are a part of the main technical program for 2011, while posters continue to provide food for thought during the Thursday evening reception. Thank you to each of the respective chairs for providing much value-added content to ACSAC: Steve Rome and co-chair Ken Shotting (case studies), Kevin Butler (panels), and Ben Kuperman (works-in-progress and posters).

ACSAC provides a tremendous venue for not only getting glimpses at innovative yet ready-to-use security work, but also gaining in-depth understanding on various security-related topics. Both the professional development courses and FISMA training track are a great way to learn a new skill or gain a deeper understanding into an area of interest. Thank you to Daniel Faigin for coordinating the courses, and to Marshall Abrams for coordinating the FISMA training as well as teaching.

New this year is Tracer FIRE (Forensic and Incident Response Exercise), a program developed by Sandia and Los Alamos National Laboratories to educate and train cyber security incident responders and analysts in critical skill areas, and to improve collaboration and teamwork among staff members. This course is split daily between classroom and competition, ensuring direct applicability of what is learned. Thank you to Ben Cook and Kathryn Hanselmann for bringing Tracer FIRE to ACSAC.

ACSAC is also privileged to welcome back two workshops – the Workshop on Governance of Technology, Information, and Policies (GTIP), coordinated by Harvey Rubinovitz, and the Layered Assurance Workshop (LAW), coordinated by Rance DeLong. These workshops provide for engaging discussions in critical areas.

Unlike many other security events, ACSAC is a non-profit conference and heavily relies on volunteers to pull this world-class event together. In addition to the volunteers named above, I would also like to express my gratitude to the following for ensuring an outstanding event: Ann Marmor-Squires, Art Friedman, Ed Schneider, Christoph Schuba, Charlie Payne, Lillian Røstad, Michael Collins, Ken Shotting, Dan Thomsen, Paul Jardetzky, and Jay Kahn. My thanks also go to the Steering Committee members who provided guidance and support, ACSA for making this event possible, and this year's sponsors for all of their contributions.

ACSAC would of course not be possible without attendees, and so I would also like to thank you for participating as well. If you are reading this prior to the event, be sure to take full advantage of the learning and networking opportunities ACSAC offers. Should you come across this message after, consider making a submission to our next event.

Robert H'obbes' Zakon, ACSAC 2011 Conference Chair

Message from the Program Chairs

Welcome to the 27th Annual Computer Security Applications Conference. We are happy to present an impressive technical program that is the work of many volunteers. We would like to thank all of these volunteers for their contributions to ACSAC 2011.

First, thanks go to the authors. We received a total of 204 submissions this year. After some submissions were rejected on formal grounds or retracted by their authors, a total of 195 submissions entered the reviewing phase.

Second, our sincere gratitude goes to the Program Committee, who gave much extra time to carefully review the large number of submissions. Papers were reviewed by three or more PC members, in a single round of review. The Program Committee meeting was held completely on-line, using the web-based OpenConf tool. The discussions intensified during the first two weeks of August, right up to the night before notifications were due to the authors. Of the 195 papers reviewed during the on-line meeting, 36 were accepted and 4 more were conditionally accepted.

Reviews were double-blinded throughout: papers were submitted without author names and affiliations and PC members never learned the identity of submitters until the reviewing process was complete. Throughout the reviewing process, the two PC Chairs were the only people who knew which paper was authored by whom. We took great care to manage potential conflicts, since in our double-blind process, reviewers were often not even aware of such conflicts.

We are happy to report that all of the conditionally accepted papers made it into the final program. Thanks go to the authors and to the shepherds for the time they collaboratively invested in improving these papers for ACSAC 2011.

And finally, we thank the larger ACSAC community for your continuing support, by submitting papers and by volunteering their time and talent in other ways. Whether you are attending ACSAC in person this year or reading these proceedings elsewhere, we hope that you will find these papers interesting, inspiring, and relevant. Enjoy!

John McDermott, ACSAC 2011 Program Chair

Michael Locasto, ACSAC 2011 Program Co-Chair

Organizing Committee

Steering Committee

Program Committee

Pierangela Samarati, Università degli Studi di Milano
Christoph Schuba, Oracle Corporation
R. Sekar, Stony Brook University
Cristina Serban, AT&T Security Research Center
Anil Somayaji, Carleton University
Angelos Stavrou, George Mason University
Bhavani Thuraisingham, University of Texas, Dallas
Patrick Traynor, Georgia Institute of Technology
Venkat Venkatakrishnan, University of Illinois at Chicago
Dongyan Xu, Purdue University
Alec Yasinsac, University of South Alabama

Professional Development Course Proposal Reviewers

Daniel P. Faigin, The Aerospace Corporation (Professional Development Chair)
Patricia Daggett, EMC Corporation
Deborah D. Downs
Richard Johnson, The Aerospace Corporation
Donna Mitchell, Lockheed Martin Corporation
W. Warren Pearce
Marco Ramilli, University of Bologna
Max Robinson, The Aerospace Corporation
Steven Rome, Booz Allen Hamilton
Michelle Ruppel, Saffire Systems
Harvey Rubinovitz, The MITRE Corporation
Cristina Serban, AT&T Security Research Center
Danielle Weigold, Lockheed Martin Corporation
Eric Winterton, Booz Allen Hamilton
Simon Wiseman, Deep-Secure Ltd

Message from the Sponsor

Applied Computer Security Associates

ACSA had its genesis in the first Aerospace Computer Security Applications Conference in 1985. That conference was a success and evolved into the Annual Computer Security Applications Conference (ACSAC). ACSA was incorporated in 1987 as a non-profit association of computer security professionals who have a common goal of improving the understanding, theory, and practice of computer security. ACSA continues to be the primary sponsor of the annual conference.

In 1989, ACSA began the Distinguished Practitioner Series at the annual conference. Each year, an outstanding computer security professional is invited to present a lecture of current topical interest to the security community. In 1991, ACSAC began the Best Paper by a Student Award, presented at the Annual conference. This award is intended to encourage active student participation in the conference. The award winning student author receives an honorarium and conference expenses. Additionally, our Student Conferenceship program assists selected students in attending the Conference by paying for the conference fee. Applicants must be undergraduate or graduate students, nominated by a faculty member at an accredited university or school, and show the need for financial assistance to attend this conference.

An annual prize for the Outstanding Paper has been established for the Annual Computer Security Applications Conference. The winning author receives a plaque and an honorarium. The award is based on both the written and oral presentations. ACSA initiated the Marshall D. Abrams Invited Essay in 2000 to stimulate development of provocative and stimulating reading material for students of Information Security, thereby forming a set of Invited Essays. Each year's Invited Essay addresses an important topic in Information Security not adequately covered by the existing literature. This year's ACSAC continues the Classic Papers feature begun in 2001. The classic papers are updates of some of the seminal works in the field of Information Security that reflect developments in the research community and industry since their original publication. Each presentation also considers how these classical security results will impact us in the years to come.

ACSA continues to be committed to serving the security community by finding additional approaches for encouraging and facilitating dialogue and technical interchange. In previous years, ACSA has sponsored small workshops to explore various topics in Computer Security (in 2000, the Workshop on Innovations in Strong Access Control; in 2001, the Workshop on Information Security System Rating and Ranking; in 2002, the Workshop on Application of Engineering Principles to System Security Design). In 2003, ACSA became the sponsor of the already established New Security Paradigms Workshop (NSPW). In 2010 we welcomed the Layered Assurance Workshop as an affiliated ACSA activity. ACSA also maintains a Classic Papers Bookshelf that preserves seminal works in the field and a web site focusing on Strong Access Control/Multi-Level Security. After last year's ACSAC tribute to Paul Karger and Bob Abbott, ACSA has initiated work on an *In Memoriam* section of our website to capture the many contributions made by our fellow practitioners. ACSA is always interested in suggestions from interested professionals and computer security professional organizations on other ways to achieve its objectives of encouraging and facilitating dialogue and technical interchange. For more information on ACSA and its activities, please visit http://www.acsac.org/acsa

To learn more about the conference, visit the ACSAC web page at http://www.acsac.org

ACSA Members

Contributing Sponsors

Adventium Labs

Alert Logic, Inc.

Booz Allen Hamilton Inc.

CA Technologies

Guidance Software Inc.

Understanding the Prevalence and Use of Alternative Plans in Malware with Network Games

Yacin Nadji
College of Computing
Georgia Institute of Technology
Atlanta, GA 30602, USA
yacin.nadji@cc.gatech.edu

Manos Antonakakis
Damballa Inc.
Atlanta, GA 30308, USA
manos@damballa.com

Roberto Perdisci
Department of Computer Science
University of Georgia
Athens, GA 30602, USA
perdisci@cs.uga.edu

Wenke Lee
College of Computing
Georgia Institute of Technology
Atlanta, GA 30602, USA
wenke@cc.gatech.edu

ABSTRACT

In this paper we describe and evaluate a technique to improve the amount of information gained from dynamic malware analysis systems. By playing *network games* during analysis, we explore the behavior of malware when it believes its network resources are malfunctioning. This forces the malware to reveal its *alternative plan* to the analysis system resulting in a more complete understanding of malware behavior. Network games are similar to multipath exploration techniques, but are resistant to conditional code obfuscation. Our experimental results show that network games discover highly useful network information from malware. Of the 161,000 domain names and over three million IP addresses coerced from malware during three weeks, over 95% *never* appeared on public blacklists. We show that this information is both likely to be malicious and can be used to improve existing domain name and IP address reputation systems, blacklists, and network-based malware clustering systems.

1. INTRODUCTION

Malware authors often implement a variety of techniques to improve the reliability of their malicious network infrastructure. For example, short-lived domain names are used by attackers to act as temporary drop sites for exfiltrated private information. Fast-flux service networks [30] let attackers rapidly change DNS resource records to improve the availability of their malicious network infrastructure. Malicious networks are becoming decentralized to eliminate a central point of failure. All of these techniques improve the reliability of malware by making them more resistant to take-down efforts. Furthermore, malware often makes use of randomness when choosing domain names or IP addresses to use to contact its command-and-control (C&C) server, which makes malware even more resilient and more difficult to analyze.

In particular, this poses a problem for dynamic malware analysis systems, which suffer from observing a single execution trace of a running program. For example, consider a malware family that randomly chooses a C&C domain name from a predefined list of 10 domains to connect to at runtime. MD5 distinct binaries of this malware family could have non-intersecting sets of domain names implying that they are unrelated samples. Furthermore, if a malware sample from this family successfully connected to the first domain name it picked and queried, we would fail to see the remaining nine domain names it could have used. These additional domain names could be instrumental in providing a malware sample a reliable way of contacting its C&C in the event some of its domain names had been remediated since it was initially created. Of existing malware samples seen in the wild, how many employ *alternative plans* in the form of additional domain names or IP addresses to contact in the event of network failure?

In this paper, we present a framework that addresses the primary weakness of dynamic malware analysis from a new point of view using the concept of *network games*. A network game tricks executing malware into revealing additional network information i.e., domain names and IP addresses. By providing fabricated network responses, malware is led to believe its C&C is unavailable forcing it to attempt to connect by whatever means necessary. Gaming malware from the network is complementary to existing approaches, such as multipath exploration [22, 38], but is resistant to evasion through conditional code obfuscation [34]. By using RFC-compliant network responses, games are *indistinguishable* from legitimate network responses from the malware's perspective on the host. Using our framework, GZA[1], we

[1]Named after the Wu-Tang Clan member also known as "The Genius"

design a suite of games to understand the use and prevalence of alternative plans in malware in the wild by exploring malware behavior under a perceived unreliable network. Furthermore, we quantify the usefulness of the additional network information to security practitioners by examining public blacklists for the appearance of this information.

The gains generated from playing network games with malware directly benefits DNS and IP reputation systems [2, 4], network-based malware clustering systems [24], and traditional domain name and IP address blacklists. Furthermore, with these games we can obtain a better understanding of domain name generation algorithms [25, 39] (DGA) to aid in reverse-engineering them.

Specifically, our paper provides the following contributions:

- We design and implement a framework, GZA, for exploratory malware analysis using the concept of network games. Network games can be implemented quickly in as little as 100 lines of Python to respond to developing malware behavior and fit the specific needs of a malware analyst.

- Using GZA, we measure the prevalence of alternative plans in malware. We analyze a large dataset containing 2,191 distinct malware samples, and show that approximately 17% of the samples exhibit alternative plan behaviors.

- We study the long-term benefits of using network games to complement existing malware analysis techniques. Network games coerce, on average, an additional three domain names and two IP addresses from samples with alternative plans. Approximately 95% of the coerced information **never** appears on public blacklists throughout the course of our study. Over 76% of the coerced domain names were flagged as potentially malicious by the domain name reputation system Notos [2].

The remainder of this paper is structured as follows. In Section 2 we explicitly define network games and the specific games we used in our study. In Section 3 we describe the implementation of GZA. We present the methodology for two studies to examine alternative plan prevalence in Section 4, with the subsequent results shown in Section 5. We discuss the implications of network games as well as potential evasion techniques and their solutions in Section 6. We discuss similarities and differences from prior work in Section 7 and conclude the paper in Section 8.

2. PLAYING GAMES WITH MALWARE

Malware uses the same network protocols that benign software uses when performing malicious activity. Despite the fact that many network protocols exist, nearly all communication on the Internet follows one of two patterns:

1. A transport layer (e.g., TCP and UDP) connection is made to an IP address directly, **or**

2. A DNS query is made for a domain name (e.g., `google.com`) and a connection to the returned IP address is made as in #1.

Higher-level protocols leverage these two use cases for nearly all communication. If we can assume malware relies on these two patterns for contacting its C&C servers and performing its malicious activities, these are the patterns we must target during analysis.

We define a *network game* to be a set of rules that determine when to inject "false network information" into the communication between a running malware executable and the Internet. More specifically, false information is a forged network packet. Consider the running malware sample m in Figure 1(a). Sample m first performs a DNS query to determine the IP address of its C&C server located at `foo.com`. The returned IP address, `a.b.c.d`, is then used to connect to the C&C and the malware has successfully "phoned home". Sample m could also bypass DNS entirely if it were to hardcode the IP address of its C&C and communicate with it directly, as we see in Figure 1(c). This gives us two opportunities to play games with sample m as shown in Figures 1(b,d): we can say the domain name resolution of `foo.com` was unsuccessful (b) or the direct connection to IP address `a.b.c.d` was unsuccessful (d). At this point, sample m has four possible courses of action:

1. Retry the same domain name or IP address,

2. Remain dormant to evade dynamic analysis and try again later,

3. Give up, or

4. Try a previously unused domain name or IP address.

In (b) and (d), we see the malware samples taking action #4 and querying a *previously unseen* domain name (`bar.com`) and IP address (`e.f.g.c`), respectively. Action #2 is a common problem in dynamic malware analysis systems in general and is further discussed in Section 6.2.

2.1 Notation

Stated more formally, let h be a machine infected with a malware sample m that is currently executing in our analysis system running game G_{name}. G_{name} is a packet transformation function called *name*. Given a packet p, $G_{\text{name}}(p)$ represents G_{name} *gaming* p and its value is either the original packet, or some altered packet p' that changes the intent of p. The implementation details of G_{name} determine when to return p or p'. For example, p could contain the resolved IP address of a queried domain name d, whereas p' says d does not exist. In all other ways, such as type of packet and source and destination IP addresses, p and p' are identical. As h communicates with the outside world, it sends question packets, q_i, in the form of domain name queries and requests to initiate a TCP connection and receives response packets, r_j, in the form of domain name resolutions and initiated TCP connections[2]. False information is provided to the host h by delivering $G_{\text{name}}(r_j)$ in lieu of r_j. A *sample set* for m, $G_{\text{name,m}}$ represents the set of unique domain names and IP addresses queried by m while running under G_{name}. The functions D and I operate on sample sets and return the subset of unique domain names **or** the subset of unique IP addresses, respectively. Given a set of malware sample MD5s, M, a *game set*, G^M_{name} represents the subset of samples that were "successfully gamed" by G_{name}. A game is considered successful if it forces a malware sample to query

[2]More accurately, a TCP response packet is a TCP SYN-ACK packet as part of the TCP connection handshake.

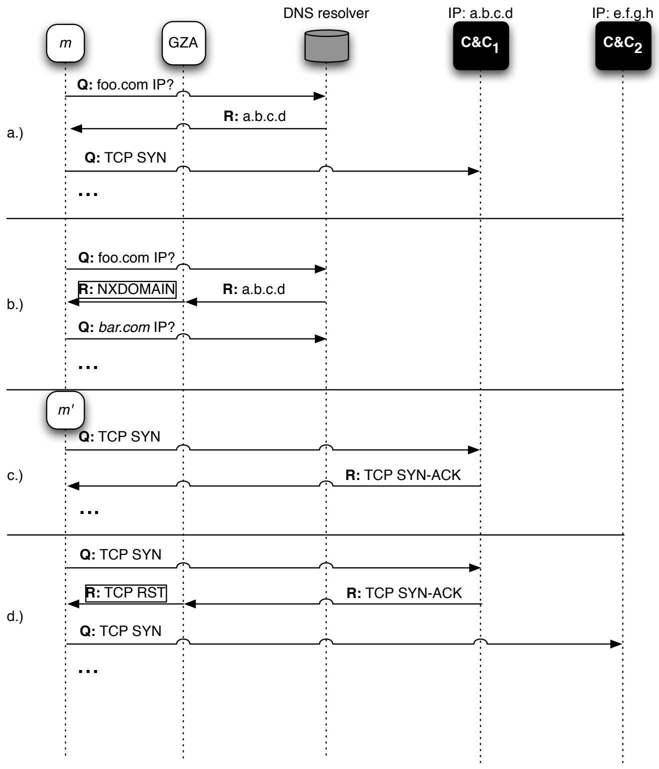


Figure 1: Malware samples m (a-b) and m' (c-d) initiating a connection with the C&C server. m connects by first performing a DNS query to determine the IP address of its C&C server followed by initiating a TCP connection. Sample m' connects directly to the C&C using a hard-coded IP address. Examples (a) and (c) connect without intervention by a game, while (b) and (d) have false information (denoted by boxes) injected.

more network information than under a run without a game present. We formally define this in the following section. The described notation is summarized in Table 1 and will be used throughout the remainder of the paper.

2.2 Designing Games for Interrogating Malware

Crafting games without a priori knowledge of malware network behavior is difficult. Furthermore, a successful game for sample m may be unsuccessful for sample m'. By using generic games "en masse", we improve our chances of successfully gaming malware during analysis. We design a suite of games to coerce a given malware sample into showing its alternative plan during analysis. We apply *all* games to a malware sample to improve the likelihood of success. Each game focuses either on DNS or TCP response packets in an attempt to harvest additional C&C domain names or IP addresses, respectively. For a DNS response packet p_d, p'_d is a modified response packet that declares the queried domain name does not exist, i.e., a DNS `rcode` of `NXDOMAIN`. For a TCP response packet p_t, p'_t is a modified response packet that terminates the 3-way TCP handshake, i.e., a TCP-RESET packet. In this paper, we choose to focus on

G_{name}	A game called *name*.
$G_{\text{name}}(p)$	The result of G_{name}'s transformation on packet p.
$G_{\text{name,m}}$	The set of network information i.e., unique IP addresses and domain names contacted, generated when malware sample m is gamed by G_{name}.
G^M_{name}	The subset of malware samples from M that were successfully gamed by G_{name}.
$D(s)$, $I(s)$	Given a sample set, s, return the subset of unique domain names or IP addresses in s, respectively.

Table 1: Notation for describing games and the sets they generate.

DNS/TCP packets as they are the predominant protocols used to establish and sustain C&C communication; however, our approach is general and can be adapted to other protocols used less commonly in C&C communication. The design of an individual game is based on anecdotal evidence of how malware samples, in general, communicate. We design seven games to perform our analysis of alternative plan behavior in malware:

G_{null}.

To provide a baseline to compare the effectiveness of future games, this game allows response packets to reach its host without modification. In other words, G_{null} is the identity function.

Note that this does not mean malware communication is allowed to run completely unfettered. We perform standard precautionary measures to prevent malicious activity from harming external systems. However, these measures are not considered part of our network games, but simply good practice when analyzing potentially malicious binaries. We discuss these precautionary measures, which are always performed irrespective of the active game, in detail in Section 3.2.

G_{dns1}.

Malware often immediately connects to its C&C or performs some probing operation to determine the status of its network before doing so. This game assumes the first domain name lookup corresponds to a test of network availability and should be allowed to pass through without modification. Subsequent domain name lookups for domains other than what was queried first will be spoofed. For example, if `google.com` is the first domain queried, all subsequent domains that are not `google.com` will be spoofed. We approximate this behavior in the next game using a whitelist. For a DNS response packet p_d, $G_{\text{dns1}}(p_d)$ returns p_d if its the first DNS request packet and p'_d otherwise. G_{dns1} is successful for m iff $|G_{\text{dns1,m}}| > |D(G_{\text{null,m}})|$.

G_{dnsw}.

A popular domain name, like `google.com`, is unlikely to operate as a C&C server for a botnet. Therefore, DNS queries on popular domain names are unlikely to be concealing additional malicious network information. For a DNS response packet p_d, $G_{\text{dnsw}}(p_d)$ returns p_d if the domain being queried is whitelisted and p'_d otherwise. Our whitelist is comprised of the top 1000 Alexa domain names [1]. G_{dnsw} is successful for m iff $|G_{\text{dnsw,m}}| > |D(G_{\text{null,m}})|$.

G_{tcpw}.
An IP address that resides in a known benign network is also unlikely to function as a C&C, much like a popular domain name. For a TCP response packet p_t, $G_{\text{tcpw}}(p_t)$ returns p_t if the IP being queried is whitelisted and p'_t otherwise. Our whitelist is the dnswl IP-based whitelist [17]. G_{tcpw} is successful for m iff $|G_{\text{tcpw,m}}| > |I(G_{\text{null,m}})|$.

G_{tcp1}.
Malware is often delivered by a *dropper*, a program that downloads, installs and runs the actual malicious binary. If we prevent the dropper from downloading its malicious payload, we will not observe the malicious behavior and fail to unearth alternative plans. We create a class of games that focus on droppers by allowing a variable number of TCP streams to successfully complete before forging response packets. For a TCP response packet p_t, $G_{\text{tcp1}}(p_t)$ returns p_t if the packet is the *first* TCP stream and p'_t otherwise. G_{tcp1} is successful for m iff $|G_{\text{tcp1,m}}| > |I(G_{\text{null,m}})|$.

G_{tcp2}.
Droppers can have multiple *stages* where malicious payloads are downloaded in more than one TCP stream. This games *two stage* droppers. This game is the same as G_{tcp1}, but allows two TCP streams to complete. For a TCP response packet p_t, $G_{\text{tcp2}}(p_t)$ returns p_t if the packet is the *first or second* TCP stream and p'_t otherwise. G_{tcp2} is successful for m iff $|G_{\text{tcp2,m}}| > |I(G_{\text{null,m}})|$.

G_{tcp3}.
While one and two stage droppers are fairly common in the wild, we wanted to test for three stage droppers. We can compare the results for G_{tcp1}, G_{tcp2} and G_{tcp3} to determine when we no longer benefit from increasing the number of allowable TCP streams. This game is the same as G_{tcp1}, but allows three TCP streams to complete. For a TCP response packet p_t, $G_{\text{tcp3}}(p_t)$ returns p_t if the packet is the *first, second or third* TCP stream and p'_t otherwise. G_{tcp3} is successful for m iff $|G_{\text{tcp3,m}}| > |I(G_{\text{null,m}})|$.

3. GZA

In this section, we describe the architecture of GZA and specific implementation details of our system.

3.1 Architecture

GZA contains two components: dynamic malware analysis and gameplay. The first component simply runs malicious code in a fresh virtual machine (VM) and records all network activity that occurs in the VM. All network activity for a VM is routed through one of the games described in Section 2.2 as seen in Figure 2. While the malware sample under analysis initiates communication with its C&C, all packets destined for a VM are routed through a network game. The game decides whether to faithfully route the response packet, or construct and send a spoofed packet, to the sample. Games are run on the host machine in isolation from the VMs, so malware cannot detect that its network activity is being analyzed and modified. As discussed earlier, all spoofed responses are RFC-compliant packets of the protocol currently being gamed making them indistinguishable from legitimate responses. As any analysis technique gains traction, malware authors will begin to attempt to circumvent it. Therefore, we discuss possible evasion techniques and how to mitigate them in Section 6.2.

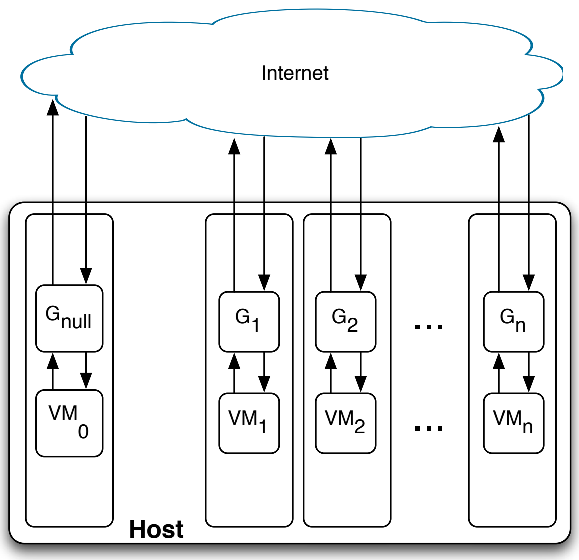


Figure 2: An overview of network traffic routing in GZA. Multiple virtual machines (VM$_i$) are run on a host using GZA. Each VM is paired up with a game G and a single sample is run against $n+1$ games. This includes G_{null} to act as a baseline. Each VM's network is isolated from all others to prevent local infection. VM network traffic is routed through its paired game to perform the required packet transformations. There can be multiple groups of these on a single host to perform bulk sample analysis.

3.2 Implementation

GZA[3] is a collection of Python scripts that run malware samples inside a virtualized Windows XP instance in `kvm` and route packets to implement the games described in Section 2.2. All applications and services that could generate DNS or TCP traffic automatically are disabled to ensure that gamed packets are from the analyzed malware only. Before packets are routed for gameplay, precautionary measures are performed to prevent malware from damaging external systems. All SMTP traffic is redirected to a spam trap to prevent spamming and traffic to local systems is dropped to prevent local infection of nearby machines or concurrently running VMs. Each VM has its packets routed through the host running `kvm` using `iptables` [23] with relevant packets being forwarded to a Python script that runs a game. This is done through the `iptables` NFQUEUE interface that redirects each packet to a user-mode process which decides if the packet should be accepted or dropped. If the game returns the original packet, the `NF_ACCEPT` message is returned to the host's kernel and the packet is routed faithfully. If the game returns a spoofed packet, `NF_REJECT` is sent to the

[3]We will make the source code for the GZA framework available in the near future.

host's kernel and a forged packet is created and sent to the VM using the packet manipulation library scapy [5].

Games are very short Python scripts that provide two external functions: `playgame` and `spoof`. `playgame` instructs the host's kernel to route the original packet or to drop it; `spoof` generates and sends a falsified packet to the VM if the original packet was dropped. GZA allows for additional games to be created and removed as its operator sees fit. The implementation of all six of the DNS and TCP games took only 113 lines of code combined.

4. STUDY METHODOLOGY

Using the idea of playing games with malware, we design and run two studies to understand alternative plans in malware. The first goal of this study is to understand the prevalence of alternative plans in malware and determine which games are the best in general; successful games force executing malware to reveal the most additional information. The second attempts to quantify how useful this previously unknown information is by determining how long it takes for newly discovered domain names and IP addresses to appear on publicly available blacklists; coerced network information is more useful the longer it takes to appear. We assume that non-whitelisted network information contacted by malware is malicious. Note that not all games use the whitelist, but during our evaluation we ignore additional network information that is whitelisted. For example, if G_{dns1} caused additional benign domains to be queried, it would not be considered successful. For TCP-based games, we also ignore additional A records returned by DNS requests. We validate this assumption by providing DNS reputation scores for domain names. Furthermore, we show how this increase in network information can improve the accuracy of network-based clustering systems. In both studies presented, all samples are run for five minutes. We discuss timing based evasion further in Section 6.2, but in short the issue is common across all dynamic analysis systems and is orthogonal to the problem we address in this paper.

4.1 Representative Study

We created a dataset, D_R, of 2,191 distinct malware samples obtained between April 2010 and October 2010 from a variety of sources, including: low interaction honeypots, web crawlers, mail filters and user submissions. We used several sources to approximate the general malware population as closely as possible. Additionally, all samples in D_R were flagged as malicious by both Symantec and McAfee. For all malware samples $m \in D_R$, we run m in GZA against each game described in Section 2.2. The astute reader will notice that we run the risk of uncovering new information by chance. Consider a malware sample that is analyzed at two distinct times, t and t' where $t < t'$. It is possible that the malicious network infrastructure changed at some time v where $t < v < t'$, which could taint our results. To eliminate this possibility, a single sample is run against all games *at the same time.*

Malware tend to rely on either domain names *or* IP addresses to communicate with their C&C. Using this assumption, we can increase the throughput of GZA for the long-term study by only using the two most successful games for each protocol.

4.2 Long-term Study

In addition to measuring the prevalence of alternative plans in malware, we want to determine how useful this information is the day a malware sample appears on a malware feed. Detecting malicious domain names and IP addresses before they have appeared in blacklists offers a tangible improvement to companies and researchers that use domain name and IP address reputation systems, perform network-based malware clustering, or maintain domain name and IP address blacklists.

Using the two games chosen from the previous study and G_{null}, we play games with malware samples provided by our daily malware feeds over the course of three weeks. Each day, we analyze all the samples we encounter on our feeds using GZA. Samples that do not generate any network traffic while executing under G_{null} are removed from our results. For each sample that is successfully gamed, we must evaluate the usefulness of the newly obtained information. To do this, we cross-reference the domain name or IP address against eleven public blacklists [12, 20, 15, 16, 21, 35, 19, 13, 27, 29, 26]. The blacklists provide two dates: d_f, the first day a domain name or IP address appeared on the blacklist and d_l, the last day a domain name or IP address appeared on the blacklist. For each additional piece of network information, n_i, and the day it was coerced, t, we place it into one of four categories:

1. `blacklisted`: if n_i appears on any of the blacklists *after* we coerced it on day t i.e., $t < d_f$.
2. `decommissioned`: if n_i appears on any of the blacklists *before* we coerced it but has since been decommissioned i.e., $d_f < d_l < t$.
3. `campaigning`: if n_i appears on any of the blacklists and is *currently* being used i.e., $d_f < t < d_l$.
4. `never`: if n_i does not appear on any of the blacklists.

Each category provides interesting insight into a malware campaign. `blacklisted` network information shows our strategy can coerce domains that other parties eventually flag as malicious. `decommissioned` network information shows that while malware may stop using a network resource, they can quickly and easily resume using one. `campaigning` network information are seen in samples that connect to multiple network resources during *normal* operation. For example, a sample randomly chooses which domain name to use to contact its C&C. `never` network information is perhaps the most interesting. These are domains and IP addresses queried by malware that *never* appear on public blacklists throughout our experiments. `blacklisted` and `never` are the most useful categories of network information and provide the best improvements to systems that rely on such information.

By comparing generated malware sets, we will extract new relationships between malware originally thought to be unrelated. Consider two malware samples, m_1 and m_2 that when run using G_{null} they query domains d_1 and d_2, respectively. However, when run using G_{dnsw}, they *both* query d_1 and d_2. m_1 and m_2 are said to be *strongly related in* G_{dnsw}. Strongly related samples have *distinct* sets of network information when run in G_{null} but *identical* sets when run in any other game. For example, consider a malware family that randomly chooses a C&C domain name to connect to at runtime. MD5 distinct versions of this malware family could have distinct game sets in G_{null} but would have identical game sets in G_{dnsw}. Strongly related samples are likely to be related in some way, for example, they could be members of the same botnet. More formally, two malware samples, m_1 and m_2, are considered strongly related in G_{i} iff:

$$C(G_{\text{i},\text{m}_1}) = C(G_{\text{i},\text{m}_2}) \text{ and } C(G_{\text{null},\text{m}_1}) \neq C(G_{\text{null},\text{m}_2})$$

where C is a function that returns either the subset of unique domain names or unique IP addresses depending on the game type of G_{i} i.e., C is either D or I from Table 1. Samples could be related without being strongly related, however, we focus only on strongly related samples in this paper. Using network games, we can improve malware clustering that uses network features. Furthermore, we use the existing domain name reputation system, Notos [2], to validate that our newly discovered domain names are actually malicious.

5. ANALYSIS

5.1 Representative Study

A summary of the results from the representative study are available in Table 2. Of the 2,191 samples in our dataset, 17% were successfully gamed by at least one of the games described in Section 2.2. Of the two types of games, DNS-based and TCP-based, G_{dnsw} and G_{tcpw} were successful the most often with 6.0% and 7.5% success rate, respectively. In most cases, the increase in network information was between one and three new domain names or IP addresses for alternative plans. A plot of network information gains is shown in Figure 3. Both graphs are heavily skewed to the right which shows that if a malware author had the foresight to include an alternative plan they used few additional network resources. For increases in IP addresses in Figure 3(a), we see little difference between each individual strategy with respect to the amount of information increase. Figure 3(b) is similarly structured, but with a large spike at 14 additional domain names for G_{dnsw}. D_R contained 37 unique samples of the same malware that all queried the same set of domain names.

In addition to understanding the successes of each game individually, examining cases where multiple games were successful on an individual sample yield insight into understanding malware alternative plans. Table 3 shows this overlap by examining pairwise Jaccard Index [36] of the game sets of each game. The large overlap of 0.93 between $G^{D_R}_{\text{dns1}}$ and the more successful $G^{D_R}_{\text{dnsw}}$ show that a naive whitelisting strategy is sufficient and improves upon hard-coding for common patterns in malware. G_{dnsw} generalizes the behavior captured by G_{dns1}. TCP-based games exhibit a much smaller overlap, primarily due to the specific staged dropper the game targets i.e., G_{tcp2} targets two staged droppers. Game performance dropped from G_{tcpw} to G_{tcp1} and G_{tcp1} to G_{tcp2}. This shows that hardcoding for droppers is less effective than a whitelisting approach.

Furthermore, the small overlap of 0.20 between all DNS-based and TCP-based gamesets, $G^{D_R}_{\text{dns}}$ and $G^{D_R}_{\text{tcp}}$ shows that malware authors focus primarily on adding reliability using additional domain names or IP addresses for their C&C servers, but rarely both. Since we can approximate our DNS-based games with G_{dnsw} and G_{tcpw} is the best performer among TCP-based games, we will use these two games in our long-term analysis.

Game	% Gamed	Min Gain	Median Gain	Max Gain
G_{dns1}	4.4%	1	2	28
G_{dnsw}	6.0%	1	3	34
G_{tcpw}	7.5%	1	2	56
G_{tcp1}	6.3%	1	1	54
G_{tcp2}	5.4%	1	1	36
G_{tcp3}	5.4%	1	1	45
Total	17.3%	-	-	-

Table 2: Summary of results of the representative study of alternative plans in malware. The most successful DNS and TCP strategies are highlighted.

	$G^{D_R}_{\text{dns1}}$	$G^{D_R}_{\text{dnsw}}$	$G^{D_R}_{\text{tcpw}}$	$G^{D_R}_{\text{tcp1}}$	$G^{D_R}_{\text{tcp2}}$	$G^{D_R}_{\text{tcp3}}$	$G^{D_R}_{\text{tcp}}$
$G^{D_R}_{\text{dns1}}$	1	**0.93**	-	-	-	-	-
$G^{D_R}_{\text{dnsw}}$	**0.93**	1	-	-	-	-	-
$G^{D_R}_{\text{tcpw}}$	-	-	1	0.50	0.45	0.46	-
$G^{D_R}_{\text{tcp1}}$	-	-	0.50	1	0.36	0.41	-
$G^{D_R}_{\text{tcp2}}$	-	-	0.45	0.36	1	0.43	-
$G^{D_R}_{\text{tcp3}}$	-	-	0.46	0.41	0.43	1	-
$G^{D_R}_{\text{dns}}$	-	-	-	-	-	-	**0.20**

Table 3: Overlap in game strategies represented by the Jaccard Index. $G^{D_R}_{\text{dns}}$ and $G^{D_R}_{\text{tcp}}$ are the union of the DNS and TCP game sets, respectively.

5.2 Long-term Study

We ran approximately 4,000 malware samples a day through GZA using three games $\{G_{\text{null}}, G_{\text{dnsw}}, G_{\text{tcpw}}\}$ from March 11th to March 31st. In general, nearly all coerced network information was never blacklisted (category `never`) during the course of our study. See Table 4 for an example of the output for a single day of analysis that took place on March 15th. Of the unique domains and IP addresses coerced, approximately 96% and 99% never appear on public blacklists by April 2nd, respectively. A small number were considered `blacklisted`, `decommissioned` and `campaigning`. A breakdown of these categories for the entire study are shown in Figure 4. As shown by the plot, almost all coerced network features never appear on public blacklists.

Network feature	Category	Count
Domains	Blacklisted	3
Domains	Decommissioned	15
Domains	In Campaign	10
Domains	Never Blacklisted	669
Domains	Total	697
IP	Blacklisted	1
IP	Decommissioned	6
IP	In Campaign	7
IP	Never Blacklisted	3381
IP	Total	3395

Table 4: Breakdown of coerced unique network information by category and protocol for March 15th.

The additional network information generated by our games makes relationships between malware samples clearer by providing a more complete picture of C&C communication. Consider a graph K where the vertices are malware samples for a given day of our long-term experiment and edges between vertices represent shared network information. For example, two malware samples that both connect to a domain name d would have an edge drawn between them in

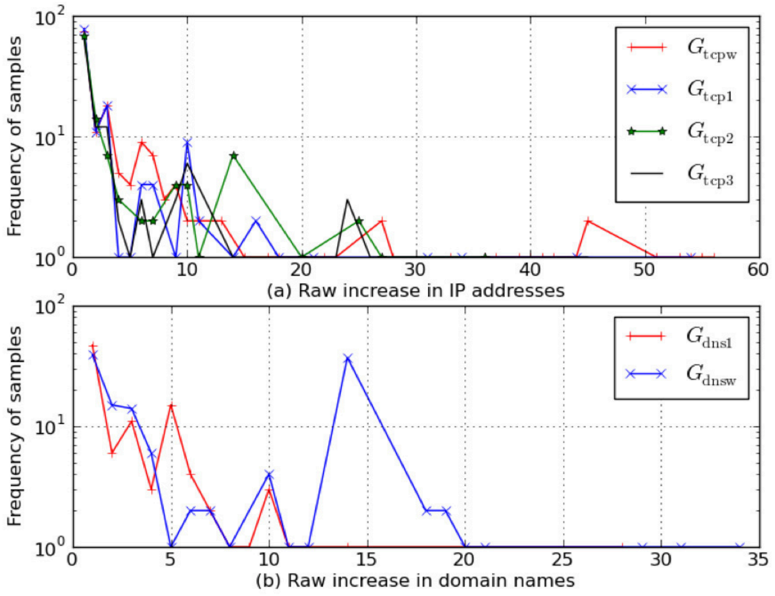


Figure 3: Plot of net network information gains for each game.

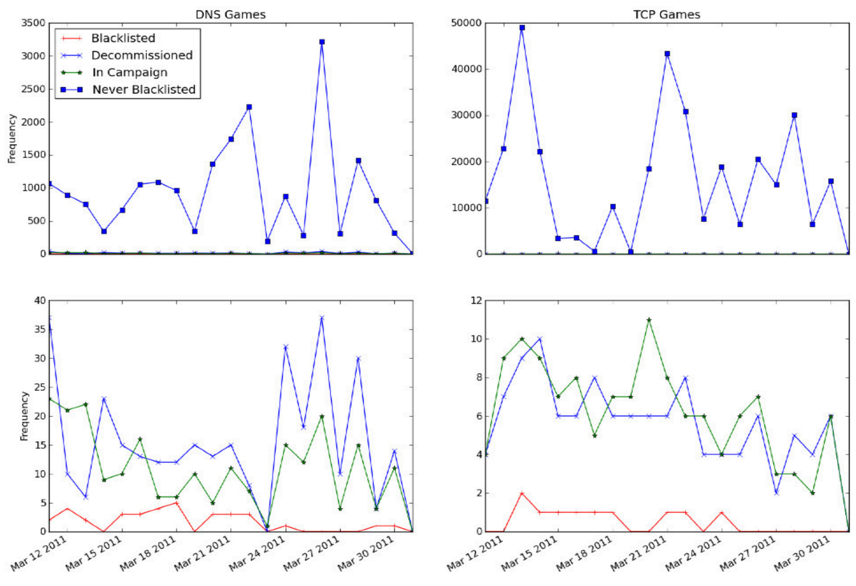


Figure 4: Frequency of network features by category over the course of the entire study. The top row of figures includes all categories, but due to the domination of the `never` **category, we also include the other three categories alone on the bottom row. Please note the change in scale.**

K. As we uncover more information through gameplay, we add additional edges into K. If these additional edges eventually form strongly related connections between malware samples, we should see a decrease in the number of components in graph K. Figure 5 shows that for G_{dnsw} we always see a drop in the number of components in the game graph of its network information compared to the graph under G_{null}. G_{tcpw} exhibited no change in the graph from G_{null}.

We also used Notos [2] to show the usefulness of our information. Given a domain name, Notos classifies the domain as: suspicious, unknown, or whitelisted. Along with a classification, Notos also provides a confidence score. Notos was trained using four weeks of passive DNS data gathered from six ISP-based DNS recursive sensors located across North America. Notos uses the top 2,000 Alexa 2LD domain names and the same blacklists used in this study. Of the 161,000 unique domain names contacted during our long-term study, we ran a simple random sample of 15,050 of them through Notos and over 76% of them were flagged as suspicious (see Table 5). The whitelisted domain names were primarily: mail servers, dynamic DNS providers, and content distribution networks. Notos had high confidence in its classification of our coerced domain names: 80% of suspicious domains and 98% of whitelisted domains had confidences above 95%.

Classification	**Count**	**Percentage**
Suspicious	11,519	76.5%
Not Known	2	$< 0.01\%$
Whitelisted	3,529	23.4%
Classification	**Mean Confidence**	-
Suspicious	0.9731[4]	-
Whitelisted	0.9956[5]	-

Table 5: Domain name classification results and mean confidence values from Notos.

6. DISCUSSION

We discuss the implications of our findings as well as potential evasion techniques malware authors may implement to circumvent network games in general.

6.1 Malware Alternative Plans

Our results report that malware is constructed naively and often relies on a single domain name or IP address to initiate and maintain a connection to its C&C server. Malware that does provide an alternative plan infrequently uses its additional network information demonstrated by the vast majority of domain names and IP addresses that do not appear on public blacklists. This may explain why only 17% of samples responded to network games. Another explanation is more specific games, perhaps dynamic ones, must be implemented to realize gains in a larger proportion of samples. We are currently exploring these questions as potential future work.

In general, coerced domain names are more useful than coerced IP addresses. Domain names and abusing the DNS allow for more volatile malicious networks than an attacker could accomplish with IP addresses alone. Furthermore, it is easier to vet the maliciousness of domain names than IP addresses, making them more attractive to security practitioners.

6.2 Evasion

Attackers are always attempting to evade newly created defenses. The most obvious ways to evade our system are through timing attacks, peer-to-peer validation of network resource connectivity, communicating with different protocols, or by evading dynamic analysis entirely with excessive timeouts prior to performing malicious behavior. We discuss these evasion techniques and present methods to address these shortcomings. Since our games use RFC-compliant network responses, malware is unable to determine if it is being gamed at the host-level and subsequently must use the network in clever ways to determine its execution environment.

Dynamic malware analysis systems generally execute malware for a fixed period of time, usually around five minutes per sample. Malware can remain dormant until this time passes to evade detection and analysis. Prior work addresses this limitation by finding these *trigger-based behaviors* and generating inputs to satisfy the triggers at runtime. This limitation applies to all dynamic analysis systems in general and is orthogonal to the problem we are trying to solve of increasing the network information an executing sample attempts to connect to.

Overhead incurred during usermode packet generation could enable a clever malware author to determine if they are being gamed or not. As a performance improvement, GZA will only route packets relevant to the game in question. For example, when G_{dnsw} is being played, `iptables` will only route UDP packets with a port of 53 destined for a VM. If DNS packets take abnormally long, while packets of other types are unaffected this could alert a malware sample that it is being analyzed. Simply routing all packets through its game would apply this overhead uniformly across all packets, removing the signal.

Peer-to-peer (P2P) evasion is when a malware sample verifies the results of a DNS or TCP request by asking another infected machine to perform an identical request. If a sample, m, cannot resolve a domain name d, but fellow infected hosts can resolve d successfully, m has reason to believe it is being run under our system. Communicating this information, however, requires the network. This forces m to succumb to gameplay one way or another; gaming of its initial C&C communication or gaming of verification queries to its peers. By focusing on the building blocks of network communication, we force all network activity to be gamed.

To perform a DNS query, a malware sample could query an HTTP-based DNS tool[6], bypassing the DNS protocol entirely. Furthermore, it could directly connect to a C&C using a non-gamed protocol, such as UDP. These problems are easily addressed by running aggregate games and adding additional protocols. Querying an HTTP-based DNS lookup tool still requires *some* network activity so running DNS and TCP games *simultaneously* would prevent this lookup from succeeding. If an attacker uses another protocol, such as UDP, it is easy to write a new game that targets this new behavior. As malware adapts to the presence of network games, malware analysts can keep pace with malware authors without too much effort.

7. RELATED WORK

Deception through gameplay has been discussed [31, 11, 8, 37], or implemented by hand [10], but little empirical work has been done to demonstrate the usefulness such an approach provides. Prior work traditionally focuses on improving information gain generated by honeypots [37, 8] us-

[6]`http://www.kloth.net/services/nslookup.php`

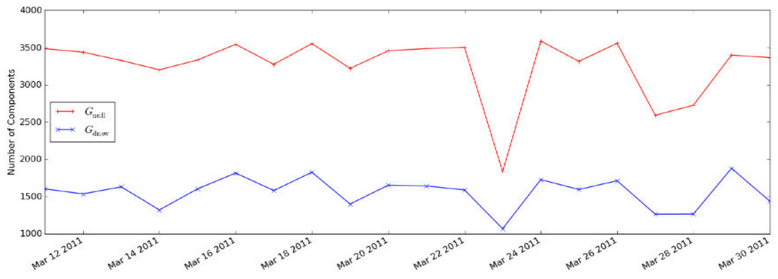


Figure 5: Numbers of components for G_{dnsw} and G_{null} for each day of the long-term experiment.

ing game theory to model interactions between an attacker and a honeypot operator. Carroll et al. focus on gains generated by having a honeypot masquerade as a normal machine, or vice-versa, and show in which cases a Nash equilibrium can be reached. Wagener et al. similarly tried to achieve equilibrium, but also played games with live intruders. The honeypot was crafted to randomly fail process spawning system calls to coerce an attacker into attempting workarounds for failing tools, hopefully leading to previously unknown tools and exploits. Gaming the botnet C&C network redundancy mechanism, what we refer to as alternative plans, was discussed and used to improve returns generated by a spamming botnet analysis engine [18]. Anticipation games [7], an extension of attack graphs which are based on game theory, were designed to anticipate malicious interaction with a network and determine the answer to questions such as determining the most effective patching strategy for a given network. We differ from previous gameplay work in that we focus on gathering network intelligence, rather than host-level information, and we quantify the usefulness of this network information to security practitioners.

GZA is similar, but complementary, to other techniques that attempt to coerce malware into revealing useful information. All systems that rely on dynamic binary analysis run into the problem of code coverage, which researchers have addressed by forcing execution of all possible branches [22, 38]. Multipath exploration provides a complete view of possible execution paths of malware but can be evaded with conditional code obfuscation [34] or made impractical due to the exponential explosion in search space. Sharif et. al. describe malware emulators [33], or malware obfuscated by a randomized bytecode instruction set, that would evade multipath exploration. During dynamic analysis, multipath exploration would explore the paths of all possible bytecode programs rather than the execution paths of the malware itself. Since network games do not target binary execution paths, we are resistant to this evasion technique and provide a complementary analysis method. Furthermore, malware increasingly uses external stimuli in the form of *trigger-based behaviors* to determine execution. Malware can determine its execution environment [32, 28, 9] prompting the use of hardware virtualization [14]. More sophisticated techniques include waiting for a specific date to occur or a particular website to be visited. Research has shown how to detect changes in malware behavior as well as determine the underlying cause [3, 6]. We differ from prior work in malware analysis by introducing the concept of evasion-resistant network games. By performing execution path exploration from the network instead of the host, we make it difficult for malware to detect it is being gamed or evade our games.

8. CONCLUSION

In this paper, we designed and built a framework, GZA, to explore malware execution paths using the concept of network games. By playing network games with malware, we described the prevalence of *alternative plans* in malware by examining a large malware corpus of 2,191 samples and performing a long-term study over three weeks of malware samples obtained from malware feeds. Our six network games coerced samples into revealing their alternative plans and the additional network features malware used to enact those plans. We show that while alternative plans have promise to be used to improve malware reliability, they go relatively unused in malware seen in the wild. Only 17% show this behavior. This new network information, however, is *very* useful with approximately 95% never appearing on public blacklists. This directly improves systems that rely on network information, such as blacklist generation, domain name and IP address reputation systems, and malware clustering on network features.

Acknowledgments

The authors gratefully acknowledge Paul Royal for providing the malware samples we used in both experiments, as well as providing advice in implementing GZA. We also thank GZA/Genius for *Liquid Swords* and *Legend of the Liquid Sword*, which were on repeat during late-night programming and editing sessions. We also thank the anonymous reviewers for their helpful comments.

9. REFERENCES

[1] Alexa. Top sites. `http://www.alexa.com/topsites`, (Retrieved) March 2011.

[2] M. Antonakakis, R. Perdisci, D. Dagon, and W. Lee. Building a dynamic reputation system for DNS. In *Proceedings of the 19th USENIX Security Symposium*, 2010.

[3] D. Balzarotti, M. Cova, C. Karlberger, C. Kruegel, and E. Kirda. Efficient detection of split personalities in malware. In *Proceedings of the Symposium on Network and Distributed System Security*, 2010.

[4] L. Bilge, E. Kirda, C. Kruegel, and M. Balduzzi. EXPOSURE: Finding malicious domains using passive

DNS analysis. In *Proceedings of the Symposium on Network and Distributed System Security*, Jan 2011.

[5] P. Biondi. Scapy. `http://www.secdev.org/projects/scapy/`, (Retrieved) March 2011.

[6] D. Brumley, C. Hartwig, Z. Liang, J. Newsome, D. Song, and H. Yin. Automatically identifying trigger-based behavior in malware. *Botnet Detection*, pages 65–88, 2008.

[7] E. Bursztein and J. C. Mitchell. Using strategy objectives for network security analysis. *Information Security and Cryptology*, Jan 2011.

[8] T. Carroll and D. Grosu. A game theoretic investigation of deception in network security. *Security and Communication Networks*, Jan 2009.

[9] X. Chen, J. Andersen, Z. M. Mao, M. Bailey, and J. Nazario. Towards an understanding of anti-virtualization and anti-debugging behavior in modern malware. In *Proceedings of the International Conference on Dependable Systems and Networks DSN*, 2008.

[10] B. Cheswick. An evening with berferd in which a cracker is lured, endured, and studied. In *Proceedings of the USENIX Security Symposium*, Jan 1990.

[11] F. Cohen and D. Koike. Misleading attackers with deception. In *Proceedings of the 2004 IEEE Workshop on Information Assurance*, Jan 2004.

[12] A. D. Correa. Malware patrol. `http://malwarepatrol.com/`, 2010.

[13] T. Cymru. Bogons. `http://www.cymru.com/Documents/bogon-bn-nonagg.txt`, 2010.

[14] A. Dinaburg, P. Royal, M. Sharif, and W. Lee. Ether: Malware analysis via hardware virtualization extensions. In *Proceedings of the 15th ACM Conference on Computer and Communications Security*, Jan 2008.

[15] DNS-BH. Malware prevention through DNS redirection (black hole DNS sinkhole). `http://www.malwaredomains.com`, 2010.

[16] dnsbl.abuse.ch. dnsbl.abuse.ch. `http://dnsbl.abuse.ch`, 2010.

[17] dnswl. DNS whitelist - protect against false positives. `http://www.dnswl.org`, (Retrieved) March 2011.

[18] J. John, A. Moshchuk, S. Gribble, and A. Krishnamurthy. Studying spamming botnets using botlab. In *Proceedings of the 6th USENIX Symposium on Networked Systems Design and Implementation*, pages 291–306, 2009.

[19] M. D. List. Malware domain list. `http://www.malwaredomainlist.com`, 2010.

[20] L. Lu, V. Yegneswaran, P. Porras, and W. Lee. BLADE: an attack-agnostic approach for preventing drive-by malware infections. In *Proceedings of the 17th ACM Conference on Computer and Communiations Security (CCS 2010)*, Jan 2010.

[21] malc0de. Malc0de DNS blacklist. `http://malc0de.com`, 2010.

[22] A. Moser, C. Kruegel, and E. Kirda. Exploring multiple execution paths for malware analysis. In *Proceedings of the IEEE Symposium on Security and Privacy*, volume 245, 2007.

[23] netfilter team. The netfilter.org "iptables" project. `http://www.netfilter.org/projects/iptables/index.html`, (Retrieved) March 2011.

[24] R. Perdisci, W. Lee, and N. Feamster. Behavioral clustering of HTTP-based malware and signature generation using malicious network traces. In *Proceedings of the USENIX Symposium on Networked Systems Design and Implementation*, 2010.

[25] H. S. Phillip Porras and V. Yegneswaran. An analysis of conficker's logic and rendezvous points. `http://mtc.sri.com/Conficker/`, 2009.

[26] S. Project. Snort DNS/IP/URL lists. `http://labs.snort.org/iplists/`, 2011.

[27] T. S. Project. Spamhaus drop list. `http://www.spamhaus.org/drop/drop.lasso`, 2011.

[28] T. Raffetseder, C. Krügel, and E. Kirda. Detecting system emulators. In *Information Security Conference*, pages 1–18, 2007.

[29] C. Report. CIDR report bogons. `http://www.cidr-report.org`, 2011.

[30] J. Riden. How fast-flux service networks work. `http://www.honeynet.org/node/132`, 2008.

[31] N. Rowe, E. Custy, and B. T. Duong. Defending cyberspace with fake honeypots. *Journal of Computers*, Jan 2007.

[32] J. Rutkowska. Red pill... or how to detect VMM using (almost) one CPU instruction. `http://invisiblethings.org/papers/redpill.html`, 2004.

[33] M. Sharif, A. Lanzi, J. Giffin, and W. Lee. Rotalume: A tool for automatic reverse engineering of malware emulators.

[34] M. Sharif, A. Lanzi, J. Giffin, and W. Lee. Impeding malware analysis using conditional code obfuscation. In *Proceedings of the Symposium on Network and Distributed System Security*, Jan 2008.

[35] spyeyetracker.abuse.ch. Spyeye tracker. `https://spyeyetracker.abuse.ch`, 2010.

[36] P.-N. Tan, M. Steinbach, and V. Kumar. *Introduction to Data Mining*. Addison Wesley, 2006.

[37] G. Wagener, R. State, A. Dulaunoy, and T. Engel. Self adaptive high interaction honeypots driven by game theory. In *Proceedings of the 11th International Symposium on Stabilization, Safety, and Security of Distributed Systems*, Jan 2009.

[38] J. Wilhelm and T. Chiueh. A forced sampled execution approach to kernel rootkit identification. In *Proceedings of the Symposium on Recent Advances in Intrusion Detection*, Jan 2007.

[39] J. Wolf. Technical details of srizbi's domain generation algorithm. `http://blog.fireeye.com/research/2008/11/technical-details-of-srizbis-domain-generation-algorithm.html`, 2008.

FORECAST – Skimming off the Malware Cream

Matthias Neugschwandtner[1], Paolo Milani Comparetti[1], Gregoire Jacob[2], and Christopher Kruegel[2]

[1]Vienna University of Technology, {mneug,pmilani}@seclab.tuwien.ac.at

[2]University of California, Santa Barbara, {gregoire,chris}@cs.ucsb.edu

ABSTRACT

To handle the large number of malware samples appearing in the wild each day, security analysts and vendors employ automated tools to detect, classify and analyze malicious code. Because malware is typically resistant to static analysis, automated dynamic analysis is widely used for this purpose. Executing malicious software in a controlled environment while observing its behavior can provide rich information on a malware's capabilities. However, running each malware sample even for a few minutes is expensive. For this reason, malware analysis efforts need to select a subset of samples for analysis. To date, this selection has been performed either randomly or using techniques focused on avoiding re-analysis of polymorphic malware variants [41, 23].

In this paper, we present a novel approach to sample selection that attempts to maximize the total value of the information obtained from analysis, according to an application-dependent scoring function. To this end, we leverage previous work on behavioral malware clustering [14] and introduce a machine-learning-based system that uses all statically-available information to predict into which behavioral class a sample will fall, *before* the sample is actually executed. We discuss scoring functions tailored at two practical applications of large-scale dynamic analysis: the compilation of network blacklists of command and control servers and the generation of remediation procedures for malware infections. We implement these techniques in a tool called FORECAST. Large-scale evaluation on over 600,000 malware samples shows that our prototype can increase the amount of potential command and control servers detected by up to 137% over a random selection strategy and 54% over a selection strategy based on sample diversity.

1. INTRODUCTION

Malware is at the root of many security threats on the internet. From spam, to identity theft to distributed denial of service attacks, malicious software running on compromised computers is a key component of internet crime. For this reason, analyzing malware and developing countermeasures against it has become an important aspect of security practice. New malware samples need to be analyzed to understand their capabilities and generate detection signatures, mitigation strategies and remediation procedures. Since tens of thousands of new malware samples are found in the wild each day, security analysts and antivirus vendors have to employ automated analysis techniques for this task.

Malware commonly employs various forms of packing and obfuscation to resist static analysis. Therefore, the most widespread approach to the analysis of malware samples is currently based on executing the malicious code in a controlled environment to observe its behavior. Dynamic analysis tools such as CWSandbox [3], Norman Sandbox and Anubis [13, 2] execute a malware sample in an instrumented sandbox and record its interactions with system and network resources. This information can be distilled into a human-readable report that provides an analyst with a high level view of a sample's behavior, but it can also be fed as input to further automatic analysis tasks. Execution logs and network traces provided by dynamic analysis have been used to classify malware samples [14, 36], to generate remediation procedures for malware infections [31] and to generate signatures for detecting a malware's network traffic [34].

One problem of dynamic analysis of malware is that it is resource-intensive. Panda Labs reported 63,000 new malware samples per day in 2010 with an upward trend [9]. Each of these samples needs to be executed, if only for a few minutes. Furthermore, it is relatively easy for malware authors to aggravate this problem by automatically generating even larger numbers of polymorphic variants. As a result of the limited analysis capacity, only a subset of the daily malware samples can be analyzed. Our own analysis sandbox, despite a large-scale, distributed deployment, has to discard tens of thousands of samples each day. This raises the question of which samples should be selected to best utilize the available resources.

Previous work on selecting samples for dynamic analysis [41, 23] has focused on diversity: that is, the goal is to determine whether a new malware sample, with a never-before-seen message digest, is actually just a minor variant or a polymorphic mutation of a previously analyzed sample. Results obtained with these techniques have shown that discarding polymorphic variants can reduce the amount of samples to be analyzed by a factor of over sixteen. The assumption behind this approach is that analyzing a polymorphic variant of a known sample will not provide any new insight, so the sample should be discarded rather than waste resources on executing it in the sandbox. Depending on the purpose for which samples are being analyzed, however, this assumption may not hold.

One motivation for operating a malware analysis sandbox is that the network behavior of malware can reveal the command and control (C&C) servers used by bot masters to remotely control infected computers. If the goal is to detect C&C servers, running multiple variants of *some* malware families can prove advantageous. This is a consequence of the constant arms race between bot masters and security professionals: bot masters need to maintain control of their bots while security professionals work to identify and take down their C&C infrastructure. As a result, the C&C servers used by a malware family change over time much faster than its code-base. Furthermore, some malware code-bases are available to multiple independent bot masters, each of which uses a distinct C&C infrastructure [15].

Rather than selecting samples for analysis based only on diversity, we therefore take a different angle and explicitly try to select for analysis those samples that will produce the most valuable analysis results. This requires us to first define how to measure the value of the output of a dynamic analysis run. We argue that this is application-dependent, and that sample selection should take into account the goals for which malware is being analyzed in the first place.

To predict whether and to what extent the execution of a sample will yield useful information, we take advantage of knowledge gleaned from samples that have already been analyzed. We extract all statically-available information on each malware sample, including structural features of the executable, antivirus detection results and the results of static classification using techniques from Wicherski [41] and Jacob et al. [23]. For all dynamically analyzed samples, we further record their behavior in the sandbox and the results of behavioral clustering using techniques from Bayer et al. [14]. Over time, our system thus assembles a knowledge-base of static and behavioral characteristics of malware samples. This knowledge-base can be mined for sample selection. We use the static information on each sample as input to a machine-learning system that aims to predict to which behavioral cluster a sample belongs, *before* we actually run the sample. We then select for execution samples expected to belong to clusters that, based on past performance, are most likely to provide useful information.

We implement the proposed techniques in a tool called FORECAST, and empirically evaluate its performance using a large collection of real-world malware binaries. Our results show that selecting samples for analysis using FORECAST can provide more useful information for a given amount of analysis resources compared not only to naive, random selection, but also to a selection strategy aimed at maximizing diversity. In summary, our contributions are the following:

- We formulate the sample selection problem as the task of choosing samples for dynamic analysis to maximize the aggregate value of the analysis results.
- We introduce novel techniques that allow us to predict the dynamic behavior of a malware sample before executing it. More precisely, they allow us to predict the behavioral cluster [14] to which the sample will belong.
- We introduce scoring functions for measuring the value of information obtained from dynamic analysis that are targeted at two practical applications; Namely the generation of network blacklists of command and control servers and the generation of procedures for the remediation of malware infections on end hosts.
- Based on these techniques, we develop a system for selecting samples for dynamic analysis according to the expected value of the information obtained from a sample's execution.
- We evaluate the proposed techniques on over 600,000 malware samples, and show that they can increase the total value of the information obtained from dynamic analysis by 134% compared to a random selection strategy and by 54% compared to a selection strategy based on sample diversity.

2. SYSTEM GOALS AND APPROACH

The goal of FORECAST is to increase the insight that can be gained from executing malware samples in an analysis sandbox, given a limited amount of computational resources. If insufficient resources are available to analyze all the malware samples that are collected each day, sandbox operators are faced with the choice of which samples to select for analysis.

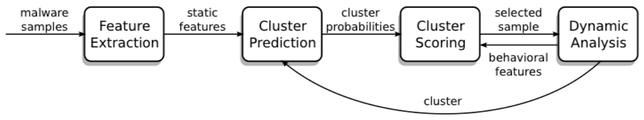


Figure 1: FORECAST **overview.**

We call this the *sample selection* problem, and formulate it as follows. Given a set χ of n malware samples, a scoring function v that measures the aggregate value of the analysis results of a set of samples and limited resources that allow the dynamic analysis of only $k < n$ samples, we want to select a subset $\alpha \subset \chi$, with $|\alpha| = k$, that maximizes $v(\alpha)$. The set α can be built incrementally: When selecting the next sample for analysis, we can take into account the analysis results for all previously analyzed samples.

Previous work [41, 23] has implicitly attempted to solve this problem by recognizing and discarding minor variants or polymorphic mutations of previously analyzed malware samples. The assumption behind this is that every such variant will exhibit the same behavior, therefore analyzing more than one variant provides no additional valuable information. Because it lacked a measure of the value of analysis results, previous work did not attempt to quantitatively validate this assumption. As we will show in Section 4, discarding minor variants is indeed a good heuristic for selecting samples. However, we will also show that in some cases executing several almost identical samples can provide valuable information, such as the different C&C servers contacted by each sample.

2.1 Applications

In this paper, we take a different approach and explicitly measure the value of analysis results. For this, we develop scoring functions targeted at two real-world applications.

Identifying C&C servers. Modern malware uses a command and control (C&C) infrastructure that allows the malware operators to remote-control the infected machines (also known as bots). It also lets them update the bots' software to adapt to the changing environment in which they operate and to the changing goals of the botnet owners. While some botnets employ peer-to-peer protocols for C&C, most employ client-server architectures and rely on redundancy and fallback mechanisms to provide robustness. The C&C servers are therefore a weak point of a botnet's operation, making information on their domain names or IP addresses extremely valuable to security practitioners. Recent research has thus focused on identifying C&C communication among the network traffic generated by malware [22]. Such information has been used for coordinated takedowns of a botnet's C&C servers, that in some cases have succeeded in completely shutting down a botnet [29]. In a few cases, C&C server information has even led to the networks of malicious internet service providers being depeered from the internet [27]. Even if the malicious servers cannot be taken down, blacklists of C&C servers such as the one provided by FIRE [39] or Zeus Tracker[8] can be used by network administrators as an additional layer of defense. A C&C blacklist provides two benefits: on the one hand, it prevents infected hosts from receiving commands that would lead them to engage in harmful behavior; On the other hand, it can alert a network administrator to the presence of infected hosts on his network. FIRE builds its C&C blacklist based on the results of large-scale dynamic analysis of malware samples with the Anubis [2] sandbox. Furthermore, the C&C traffic captured during malware execution can be used to automatically generate detection signatures [34], that can be deployed on network intrusion detection solutions. The *network endpoints* scoring function discussed in Section 3.4 is therefore designed to measure the number of potential C&C servers observed during analysis.

Generating remediation procedures. When malicious code obtains unrestricted execution privileges on an infected host, completely reinstalling the affected machine is typically the only sound way of guaranteeing that the malware is fully eradicated. For a given, known malware, however, it may be possible to generate a reliable remediation procedure that is able to revert the effects of the malicious code on the system and avoid the cost of reinstallation. Remediating malware infections is a task routinely performed by anti-virus software, with varying levels of success. Recent research has proposed techniques for automatically generating such remediation procedures [31]. These techniques are based on dynamic analysis: malware samples are executed in an instrumented environment, and all persistent modifications of the system state are recorded. A remediation procedure essentially consists of a list of affected system resources, that have to be reset to a clean state. While the techniques in [31] include methods for generalizing the observed behavior to some extent, it is clear that behavior that was never observed cannot be remediated. To provide a more complete set of remediation procedures, it is therefore desirable to observe the widest possible variety of system-modifying behavior. The persistent modifications scoring function discussed in Section 3.4 is therefore designed to measure the amount of distinct system resources affected by malware execution.

2.2 System overview

Figure 1 shows a high-level overview of FORECAST's architecture. FORECAST works in four phases:

Feature extraction. For each sample that is being considered for analysis, we first extract a number of *static features*. These features represent all the information we can efficiently obtain about a malware sample without executing it. We consider a wide variety of static features. First of all, we extract a number of structural features about the malicious executable. We consider information on the origin of the malware sample, such as the user responsible for its submission to the analysis sandbox. We also include detection results from a number of anti-virus engines. Finally, we leverage previous work on detecting polymorphic malware variants and include a sample's peHash [41], as well as its static cluster and packing level obtained using techniques from Jacob et al. [23].

Cluster prediction. The dynamic analysis phase (discussed below) identifies the behavioral cluster to which each executed sample belongs. Together with the static features, this serves as input to the cluster prediction phase. Here, we attempt to predict to which behavioral cluster a sample belongs, using a supervised learning approach. For this, we use a confidence-weighted linear classifier that outputs the probabilities that a considered sample belongs to each behavioral cluster. Whenever a sample is dynamically analyzed and assigned to a cluster, our classifier is updated to account for this new information. Note that, while cluster prediction uses supervised learning, the behavioral malware classification techniques we employ are unsupervised [14], and new behavioral clusters are added incrementally as they are discovered.

Cluster scoring. In this phase we measure the *cluster score* for each behavioral cluster C, defined as the average contribution of a sample in that cluster to the scoring function $v(C)$. This step is therefore dependent on the choice of an application-specific scoring function. The scoring function takes as input the *behavioral features* observed during the execution of each sample, and measures their aggregate value. We employ two scoring functions targeted at the applications discussed in Section 2.1. For each sample that is considered for analysis, we can then compute its expected contribution to v based on the cluster scores and the cluster probabilities obtained in the previous phase. We call this the *sample score*. The output of the cluster scoring phase is the highest scoring sample, out of a pool of candidates, that is passed to the dynamic analysis phase. Whenever the dynamic analysis results for a sample become available, the cluster scores and sample scores are incrementally updated.

Dynamic analysis. Once a malware sample has been selected, we analyze it by running it in our instrumented sandbox. The host-level and network-level behavior observed during execution is condensed into a set of behavioral features. This set is fed back to the cluster scoring phase. Furthermore, we cluster all analyzed samples based on the set of behavioral features they exhibit, using techniques from Bayer et al. [14]. The behavioral cluster to which each analyzed sample belongs is fed back to the cluster prediction phase.

3. SYSTEM DESCRIPTION

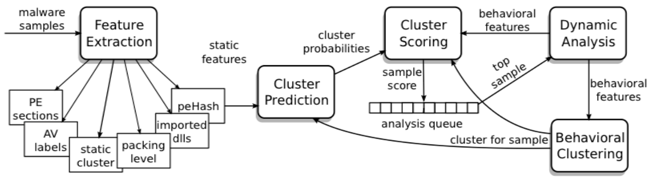


Figure 2: FORECAST **architecture.**

Figure 2 provides a more detailed view of FORECAST's architecture. FORECAST takes as input a set χ of *candidate* malware samples. The system's goal is to select for analysis a subset $\alpha \subset \chi$. FORECAST works incrementally: when selecting the next samples for analysis, it takes into account the analysis results for all previously analyzed samples. For this, FORECAST maintains an analysis queue where each not-yet-analyzed candidate sample (that is, each sample in $\chi \backslash \alpha$), is associated with a *sample score*. The sample score is a measure of how much valuable information we expect to obtain from the analysis of that sample. Of course, to achieve high throughput, malware analysis sandboxes need to analyze several samples in parallel. Thus, at each iteration FORECAST selects the L top-scoring samples for analysis, where L is the parallelism level of the sandbox, indicating the number of samples that it is able to analyze in parallel.

3.1 Dynamic analysis and clustering

Each sample selected for dynamic analysis is executed in an instrumented sandbox environment for a fixed amount of time (currently four minutes). The output of dynamic analysis is the set of behavioral features β_s observed during the execution of s. Behavioral features are a representation of program behavior introduced in [14]. Each behavioral feature represents a specific action performed by the analyzed program on a specific operating system or network entity, such as the creation of file `C:\system32\svcshost.exe`, or an HTTP request to `www.example.com`. In the behavioral clustering phase, β_s is fed to the scalable behavioral clustering techniques from Bayer et al. [14]. As a result, we identify the behavioral cluster C_s to which s is assigned.

The cluster prediction phase receives the label C_s and incrementally updates its classifier. Furthermore, β_s and C_s are fed back to the cluster scoring phase, where the *cluster score* for cluster C_s is updated based on the value of the observed behavior β_s with respect to the application-specific scoring function v. As a result, the sample scores for all samples are incrementally updated, and the next sample can be selected for analysis.

3.2 Feature Extraction

The goal of the feature extraction phase is to collect all the information on malware samples that can be used to classify it. Therefore, in this phase we extract all the characteristics of each malware sample that can be efficiently obtained from static analysis. To distinguish them from the behavioral features resulting from the dynamic analysis of a sample, we refer to these characteristics as *static* features. The output of this phase is, for each candidate sample s, a set of static features ϕ_s. The feature space $\Phi = \cup_{s \in \chi} \phi_s$ grows as new candidate samples are processed and new static features are discovered.

peHash. Unlike previous approaches to sample selection, FORECAST assumes that minor variants of previously analyzed malware samples are worth analyzing. Knowing that a candidate sample is similar to previously analyzed samples can be extremely useful when attempting to predict its behavior. Therefore, we include a sample's peHash [41] in the static features. peHash uses structural information from an executable file's headers and the Kolmogorov complexity of code sections to compute a hash value that should remain constant across polymorphic malware variants.

Static cluster. Similarly, we take advantage of static malware clustering techniques from [23]. These techniques compute the distances between the code signals – essentially a bigram distribution over a code section – of binaries to group them into clusters. The authors also introduce techniques to statically detect the level of packing used by a sample. We call these clusters *static* clusters. Both of these characteristics – *packing level* and *static cluster* membership – are mapped to individual static features.

PE Header. A variety of information from a binary's Portable Executable (PE) header is already processed by peHash and [23], and is thus already represented in the feature space. We extract two additional groups of features from the PE header. These are the names of the *imported DLLs* and the *PE section names*.

Antivirus Labels. We obtain anti-virus results for all candidate samples from the VirusTotal service [7]. VirusTotal scans each sample using 39 AV engines, and for each detection provides the name of the engine that detected the sample and the label it assigned to it. To avoid an excessively large number of AV label features we discard the most specific part of the AV labels (indicating the malware variant) and normalize the labels to a canonical form – e.g. `Trojan/Downloader.Carberp.n` becomes `carberp`.

Submitter Information. For each sample that is submitted to the sandbox, the hostname of the machine it has been submitted from is logged. Depending on how a malware analyst collects samples and on the types of malware she is interested in, the samples she submits may be skewed towards specific malware classes. We therefore map each distinct hostname from which a sample was submitted to a static feature.

3.3 Cluster Prediction

The goal of the cluster prediction phase is to predict to which behavioral cluster a candidate sample belongs, before executing it. Cluster prediction therefore aims to establish a mapping between a sample's *static* features ϕ_s and the behavioral cluster C_s to which it will be assigned based on its *behavioral* features. For this task, we take a supervised learning approach and train a classifier by providing it with a labeled dataset consisting of (ϕ_s, C_s) for each sample s that has already been subject to dynamic analysis and clustering. For this, we require an appropriate classification algorithm, considering our requirements and the properties of our datasets:

- High-dimensional feature space: $|\Phi|$ can be very large.
- Sparse data: Every sample only exhibits a small subset of the possible features ($|\phi_s| << |\Phi|$). Individual features may occur infrequently.
- Incremental operation: The results of dynamic analysis and clustering should have an immediate effect on following predictions. Furthermore, the size of the feature space and the number of clusters may change over time.
- Fast prediction: We need prediction results on all candidate samples before we can select the best ones for analysis.

For FORECAST we use linear classification in combination with the confidence-weighted (CW) learning scheme of Dredze, Crammer and Pereira [18].

Linear classification. A set of static features is represented as a binary feature vector $\vec{x}$, where every possible feature is either present (one) or absent (zero) for a specific sample. A linear classifier determines a margin y for a given feature vector $\vec{x}$ by computing the scalar product with a weight vector $\vec{w}$: $y = \sum_i x_i w_i$. The linear classification process can be visualized as splitting the feature space into two sections with a hyperplane that is determined by the weight vector $\vec{w}$, where the sign of the margin y tells us on which side of the hyperplane an input vector is, thus distinguishing two classes. The absolute value of the margin $|y|$ can be interpreted as the confidence in this classification.

Linear classifiers can handle high dimensional feature spaces and perform especially fast on sparse binary feature spaces. This is because for binary spaces, the computational cost of computing the scalar product $\vec{w} \cdot \vec{x}$ is proportional to $|\phi|$, rather than to the total number of features in the feature space $|\Phi|$.

Confidence-weighted learning. The effectiveness of a linear classifier depends on the algorithms used to train it. In online learning, the training instances are supplied one after the other. Training algorithms for linear classifiers use update rules that adjust the weights at each iteration t based on the current weights and a function g of the training instance features and label: $\vec{w}_{t+1} = \vec{w}_t + g(\vec{x}_t, y_t)$. If the feature space grows, weights can simply be added to the classifier.

Sparse data may pose a problem to such an algorithm. The reason is that typically weights are updated only if the corresponding feature is set in $\vec{x}$. Therefore, the weights for features that occur seldom are based on much less information than weights for frequently-occurring features. CW learning addresses this problem by maintaining confidence information for each feature depending on how often it occurred in training. Confidence is modeled with a Gaussian distribution $\mathcal{N}(w_i, \Sigma_i)$ for each feature i. The larger Σ_i is, the smaller is confidence in w_i and the more aggressively the distribution will be updated by g. For a full discussion of this classifier and the update function g we refer the reader to [18].

Multiclass prediction. The linear classifiers we have discussed so far can distinguish two labels. FORECAST however needs to predict to which of several behavioral clusters a sample belongs. Such a multi-class problem can be decomposed into multiple binary problems. These are then solved by a network of n binary classifiers. As a last step, a single multi-class label has to be derived from the n binary results. For this, we can use Error Correcting Output Codes (ECOC) [10]. Considering the results of the n classifiers as an n-length binary codeword, the distances between the codeword from the classification and the codewords of each label are calculated. The prediction result is the label for which the distance is minimized.

One way of performing multi-class classification using ECOC is to use a binary classifier for each pair of labels, where each classifier is trained to distinguish between these two labels. This approach is not applicable for FORECAST, because it would require us to train $|C|^2/2$ classifiers. With over a thousand clusters, this is clearly problematic. This approach is also referred to as pairwise coupling [20] or one-versus-one (OVO) classification. Instead we use a one-versus-all (OVA) approach, which requires only a single classifier per label. Each classifier is trained to distinguish between its assigned label and the "rest", i.e. all other labels. For each training instance the binary classifier of the correct class is trained with $+1$ and all other classifiers with -1.

FORECAST's cluster prediction uses one-versus-all CW learning. An advantage of this choice is that, whenever a new behavioral cluster C emerges from the analysis results, we can add a new classifier and train it to recognize C. For this, we train the new classifier with all past samples. The classifiers for all other samples, however, do not need to be modified. The weights used for classification are stored in a matrix w, where $w_{C,i}$ is the weight of feature i for cluster (classifier) C.

Probability estimates. Just picking the top predicted cluster and proceeding with the cluster scoring would mean to discard important information – the confidence in the prediction, given by the margins of each classifier. Based on that output it is possible to calculate probability estimates for each label. For ECOC models, a generic approach is given in [21]. Here we use a simple approach from the LIBLINEAR project [19] for OVA classification. The probability estimate for label C is computed as an exponential function of the margin y_C of the corresponding classifier $\frac{1}{1+e^{-y_C}}$.

Cluster size threshold. As we will show, the clusters in our behavioral malware clustering vary a lot in terms of size, with a few very large clusters and a large number of clusters containing only a few samples. Clearly, we do not want to train tens of thousands of OVA classifiers to recognize clusters that contain only one sample. First of all, this would significantly slow down FORECAST. Furthermore, a classifier that was provided during training an extremely small number of positive training instances would be unlikely to provide good results. Therefore, we select a minimum cluster size threshold θ and group all samples belonging to clusters with $|C| < \theta$ in a single "other" cluster O. We do not, however, train a classifier for O. The reason is that samples in O have nothing in common and thus their static features will vary wildly. As a probability estimate for O we therefore simply use the ratio $|O|/|\alpha|$ of the number of samples in O to the total number of analyzed samples.

As a final step all probability estimates are normalized so that they add up to one. The output of the cluster prediction phase is the cluster probability matrix p, where $p_{s,C}$ represents the probability that sample s belongs to behavioral cluster C.

3.4 Cluster Scoring

Given a measure of the value of the analysis results produced by the sandbox, we can evaluate which behavioral clusters produce more valuable results, and try to select analysis samples that will likely fall into those clusters. For this, we calculate a *cluster score* for each behavioral cluster.

As discussed in our formulation of the sample selection problem in Section 2, the value of analysis results is measured by an application-dependent scoring function $v(\alpha)$, which computes the aggregate value of the analysis results for the set of analyzed samples α. As discussed in Section 3.1, the results of dynamic analysis of a sample s is the set β_s of behavioral features observed during the sample's execution. The scoring function is therefore calculated as $v(\alpha) = v(\{\beta_s : s \in \alpha\})$. Depending on the target application, only a subset of the observed behavioral features may be of interest. For instance, if the goal of analysis is to identify malware C&C servers, only features related to network traffic are relevant, while features representing interaction with the local system can be ignored. The scoring functions we considered in this work simply measure the total number of relevant behavioral features observed. For this, each scoring function has an associated filter $f(b) \rightarrow \{0, 1\}$ that returns 1 if feature b is relevant. We then compute $v = |\{b \in B : f(b) = 1\}|$, with $B = \bigcup_{s\in\alpha} \beta_s$. Note that assigning different weights to different behavioral features, to express the fact that some features may be more valuable than others, would be a straightforward generalization of this approach.

In this work, we use the following two scoring functions, targeted at the applications discussed in Section 2.1.

Identifying C&C servers To assist in the task of identifying and blacklisting C&C servers, we introduce the *network endpoints* scoring function. This scoring function aims to measure the number of potential C&C servers contacted during dynamic analysis. The features relevant for C&C server detection are those indicating network communication with an IP address, and those indicating a DNS request for resolving a domain. A simple scoring function could therefore count the number of distinct network endpoints (IP addresses and DNS names) contacted by the analyzed samples. However, not all network traffic observed is related to C&C, or to malware's auto-update functionality (which can be seen as a type of C&C where commands are delivered in the form of executable code). Therefore, we employ a number of additional filters to make the network endpoints scoring function a more reliable measure of the amount of C&C servers contacted.

- *fast-flux*: C&C infrastructure is sometimes hosted on fast-flux networks, where the same domain name resolves to a rapidly-changing set of IPs. These IPs typically belong to infected hosts that temporarily serve as C&C servers. In this case, the information on the contacted IP addresses is of little value. Therefore, we consider IP addresses for the network endpoints score only if the malware did not obtain them from a DNS query.
- *portscan*: Many malware samples include self-propagation components that scan the internet for targets before attempting to compromise them. Clearly, we do not want to include these hosts in the network endpoints score. For this, we first discard connections to a small number of ports that are known to be typical exploit targets, such as TCP ports 139 and 445, used by Windows file and directory sharing services. Furthermore, we use the Bro IDS [32] to detect port and address scans, and discard connections that are part of detected scans.
- *liveness*: C&C endpoints that are not actually available are of little interest. We therefore filter out endpoints that could not be successfully contacted by the malware. To decide on whether a C&C endpoint is live, we rely on the known semantics of a few protocols commonly used for C&C, and fall back to a default heuristic for other kinds of traffic:
  - HTTP: we only consider servers that responded with a successful status code (that is, 200-299) to at least one request.
  - IRC: we only consider a server if the malware sent or received a private message or if it successfully joined a channel and sent or received a message on that channel.

- FTP: we only consider a server if the malware successfully logged in.
- Other: we only consider endpoints where a connection was successfully established (for TCP) and where the server sent back actual payload.

- *clickbots*: Clickbots are malware samples that include functionality to automatically visit advertisement links on target websites. Their goal is to fraudulently generate advertisement revenue for these websites. Due to the dynamic and tiered nature of advertisement networks, this can lead to a significant number of network endpoints being contacted during analysis. Since these endpoints are not related to command and control, we filter most of them out by using an existing list of ad-related domains that is manually maintained for the purpose of blocking advertisement [1].

Generating remediation procedures. To assist in the task of generating remediation procedures for malware infections, we introduce the *persistent changes* scoring function. This scoring function measures the number of distinct system resources affected by malware execution. As in [31], we take into account modifications to the file system and the windows registry. Furthermore, we also consider the processes and services started by the malware, because remediation procedures, if they need to be applied to a running system, will need to make sure that the malicious code is not running. Each system resource is identified by its name. The persistent changes score therefore counts the total number of distinct names of file and registry keys that are created or modified, as well as the processes and services started. Furthermore, our dynamic analysis phase uses techniques from Bayer et al. [14] (that are based on dynamic taint analysis) to detect randomly generated file names, and replace them with a special token. Likewise, we detect and replace names that are obtained by enumerating directories and registry keys. The idea is that a malware that generates a different random or temporary file in each execution, or one that crawls the entire file-system, infecting all the executables it encounters, should not be assigned a high persistent changes score because of this.

Once a scoring function is selected, we can measure the total value $v(C)$ of the information provided by the analysis of samples in a cluster C. We then calculate $cluster\_score_C = v(C)/|C|$. The cluster score is thus a value to cost ratio: The amount of valuable information provided by the cluster, divided by the analysis resources that have been spent to obtain it.

To calculate the sample scores for a sample s, we proceed as follows. Rather than simply consider the most likely cluster, we take into account the entire cluster probability vector p_s. The sample score is therefore the expectation of the cluster score, calculated as the scalar product $sample\_score_s = p_s \cdot cluster\_score$. At each iteration of FORECAST, the L candidate samples with the highest sample score are passed to the dynamic analysis phase.

3.5 Online Operation

As a result of the analysis of a selected sample s, the set of observed behavioral features β_s becomes available. These in turn are passed on to the behavioral clustering phase, which determines the cluster C_s. This newly obtained information then needs to be incorporated into our knowledge-base. Thus, we update the cluster score for cluster C_s. Furthermore, the cluster label C_s for the newly analyzed sample is fed back to the cluster prediction phase for learning. Therefore we update the classifier's weight matrix w.

At the next iteration of FORECAST, before selecting the next L samples, we recompute the probability matrix p for all remaining candidate samples, and recompute their sample scores. For this, we need to repeat the prediction step for each sample in $\chi \setminus \alpha$. The actual size of χ depends on how FORECAST is deployed in practice. Our simulation results in Section 4.3 are obtained using one day of malware samples from a large-scale sandbox deployment as candidate set. In this scenario, χ can contain tens of thousands of samples. Thus, this step is one of the more computationally expensive in FORECAST's operation. However, we only need to perform this step once every L samples. As we will show in Section 4.4, for realistic levels of parallelism this leads to more than acceptable performance.

4. EVALUATION

To develop FORECAST, we used a dataset of malware samples analyzed by an analysis sandbox in the months of October, November and December 2008. The resulting dataset consists of 100,408 samples. To evaluate FORECAST we use a larger, more recent dataset, that includes all the samples analyzed in the months of July, August and September of 2010. This 2010 dataset consists of 643,212 samples.

Table 1 shows an overview of the distribution of static features among feature groups for the 2010 dataset. We can see that the feature-space is large, with over 700 thousand distinct features. However, the mean number of features for each sample is only 21. For several of the feature groups, such as the peHash or static clusters groups, each sample is in fact assigned exactly one feature.

Table 1: Distribution of features in the 2010 dataset.

Feature group	Total	Average
PE section names	42389	4
Imported DLLs	12386	5
peHash	218380	1
Static clusters	203078	1
Packing Level	4	1
AV labels	250852	8
Submitter information	1909	1
Total	729007	21

We determined FORECAST's parameters by testing our system on the 2008 dataset. For the behavioral clustering, we use the same parameters and distance threshold employed in [14]. Recall that FORECAST trains a OVA classifier for each cluster C such that $|C| \geq \theta$. The samples in smaller clusters are instead assigned to the other cluster O. For our experiments, we selected $\theta = 20$. Decreasing θ beyond this point leads to a slow decrease of $|O|$, while causing a sharp increase in the number of behavioral clusters.

4.1 Cluster Prediction

Table 2: Contribution of feature groups to cluster prediction accuracy for the 2010 dataset

Feature group	Prediction Accuracy	
	Using all but FG	Using only FG
None	68%	-
PE section names	67%	45%
Imported DLLs	66%	40%
peHash	66%	45%
Static clusters	66%	45%
Packing level	68%	-
AV labels	65%	52%
Submitter information	68%	-

To assess the accuracy of FORECAST's cluster prediction, we train FORECAST's classifier on the samples from the first month and measure its accuracy on those of the last two months. Since

FORECAST is incremental and adapts with every sample that is analyzed, a dedicated training set is not strictly necessary. However, it is still desirable to bootstrap the system on some initial data, so that predictions can be based on reasonable knowledge. For the 2010 dataset, the training set of July 2010 consists of 193,726 samples while the testing set of August and September 2010 includes 449,486 samples. Samples are processed for prediction and training in chronological order.

The 2010 dataset includes a total of 1303 behavioral clusters, including the other cluster O. The first line of Table 2 shows the cluster prediction accuracy when using all static features. For 68% of the 449,486 samples, our classifier assigned the highest probability to the correct behavioral cluster (out of the 1303 behavioral clusters). The following lines show the classifier's accuracy if it is trained *without* features from one of the groups listed in Table 1 (left column), and if it is trained using *exclusively* features from a single group (right column). Table 2 thus provides some insight into the contribution of each feature group to the classifier's prediction accuracy. We can see that removing any single feature group does not cause large drops in accuracy. The reason is that features from the different groups are highly correlated. For instance, peHash and the static clustering from [23] lead to similar classifications of malware binaries. Therefore, removing the peHash features causes only a modest decrease in accuracy, because the static features provide similar information. Likewise, samples in each static cluster are typically assigned only a handful of different AV labels. Nonetheless, the right column shows that none of the feature groups, on their own, are sufficient to obtain comparable classification accuracy.

4.2 Cluster Scoring

The scoring functions proposed for FORECAST are designed to assist the selection of samples for specific analysis goals, by measuring the value of the information provided by an analysis run. It is important to verify that the proposed scoring functions indeed encourage the selection of samples that are relevant to the analysis goals. For this, for each scoring function, we rank the 1,303 clusters in the 2010 dataset by their cluster score $v(C)/|C|$, and manually assess some of the highest- and lowest-scoring clusters, as well as some of the largest clusters in the dataset. For space reason, we present results only for the network endpoints scoring function. Results for the persistent changes scoring function are available in an extended version of the paper [30].

The network endpoints scoring function is designed to encourage the analysis of samples that are likely to reveal new C&C servers. Therefore, we would expect the highest-ranked clusters for this score to belong to bots and other remote controlled malware, and especially to malware families that use a highly redundant or dynamic C&C infrastructure. Table 3 shows significant clusters of the dataset ranked by the network endpoints score. Here, the feature count is the total number of potential C&C servers detected in this cluster.

Indeed, almost all top-ranked clusters belong to remote controlled bots or trojans. Furthermore, several of these are associated with malware families that are known to use highly dynamic C&C infrastructure. One example is the Pushdo/Cutwail botnet (discussed extensively in [17]), found at rank three. Pushdo binaries contain a frequently-updated list of IP addresses. The malware contacts these addresses over HTTP to download a binary payload: typically an instance of the Cutwail spam engine. Cutwail then proceeds to obtain templates and instructions for spamming from other C&C servers over a custom binary protocol. Likewise, the Koobface botnet, at rank 28, is known to use compromised web pages as part of its C&C infrastructure. Bredolab, at rank 15, is a downloader similar to Pushdo [38].

Vundo/Zlob at rank eight, is a Fake-AV downloader. These samples download binaries from a number of different servers. This is a representative example of cases where a diversity-based selection strategy would cause the analysis sandbox to miss relevant behavior. The reason is that the 26 samples in this cluster have only two distinct peHash values. Discarding the remaining 24 samples would cause most of the 19 C&C servers to remain undiscovered.

In a few cases, however, we assign a high network endpoints score to samples that we are not necessarily interested in. The top non-relevant cluster is the Adrotator cluster at rank six. The samples in this cluster are clickbots: They visit advertisement links to fraudulently generate advertisement revenue. In this case, some of the advertisement servers in questions are not in our adblock blacklist, therefore they contribute to the network endpoints score.

A total of 171.533 samples belong to clusters with score zero. Among these an Allaple cluster can be found, which performs network scans that are filtered by the scoring function as well as another cluster performing no network activity at all.

Table 3: Selected clusters ranked by network endpoint score

Rank	Size	Feature Count	Malware Family
1	36	84	Unknown Bot
2	20	20	Harebot
3	29	26	Cutwail
...			
6	31	24	Adware Adrotator
8	26	19	Vundo / Zlob
15	67	32	Bredolab
28	141	51	Koobface
36	82	27	Swizzor
67	173	33	zBot
199	121272	4027	"other" Cluster
273	189285	2596	Unknown Downloader
...			
Last	16370	0	Allaple/Rahack
Last	28179	0	No activity

4.3 Simulation

To assess the real-world impact of performing sample selection using FORECAST, we perform a trace-based simulation. For this, we consider all of the samples that our sandbox was able to analyze during August and September 2010, and simulate a sandbox deployment with a smaller amount of resources, that is therefore able to analyze only a specified percentage of these samples. This allows us to compare FORECAST with other sample selection strategies, and measure the effect of each strategy on the value of the analysis results. The fact that our existing sandbox is considered, for the purpose of this simulation, to have 100% capacity does not mean that it is in reality able to process all available samples. However, we clearly cannot include in our evaluation samples for which we do not have dynamic analysis results. To simulate a FORECAST deployment, we split up the last two months of the 2010 dataset according to the day on which each analysis result was produced. For each day, we then perform sample selection using five different strategies, taking into account information obtained from sample analysis during the previous days.

- *Random*: Randomly select the samples for analysis. This simple selection strategy provides a baseline against which other strategies can be evaluated.
- *PeHash*: This selection strategy uses a sample's peHash [41] to attempt to maximize the diversity of the analyzed samples. For this, we randomly select a sample for analysis for each distinct peHash.

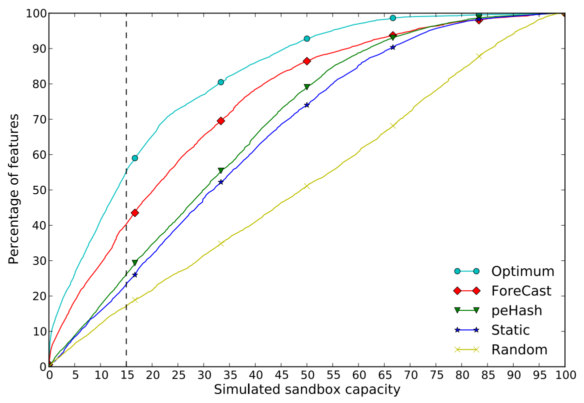


Figure 3: Daily simulation results for network features, $L = 100$

Once all these samples have been analyzed and if more analysis resources are available, we proceed to select a second sample for each peHash, and so on.

- *Static*: This selection strategy is similar to the previous one, but uses a sample's static cluster, identified with techniques from Jacob et al.[23], instead of its peHash. Together with the peHash selection strategy, this represents the state of the art in sample selection.
- *ForeCast*: We select samples for analysis using FORECAST. We test FORECAST with values of L (the level of parallelism) ranging between 50 and 1600.
- *Optimum*: Here we perform sample selection based on FORECAST's cluster scoring technique, but assuming that cluster prediction is 100% accurate; That is, that we know to which behavioral cluster each sample belongs, *before* executing it. Clearly, such a selection strategy is not possible in practice, but it serves as an upper bound on the benefits a sample selection strategy can bring.

Figure 3 shows the simulation results for the network endpoint scores and $L = 100$. On the X-axis, we have the capacity of the simulated sandbox relative to the real sandbox. This is the percentage of the samples from each day that the simulated sandbox is able to handle. On the Y-axis, we have the percentage of relevant features observed over the entire 61 day period. Thus, the Y-axis represents the percentage of C&C endpoints discovered by the simulated sandbox.

Table 4: Simulation results at 15% sandbox capacity.

	Number of features	Percentage of features	Impr. over random
Random	1645	17%	0%
Static	2242	23%	36%
peHash	2504	26%	52%
FORECAST, $L = 50$	3900	41%	137%
FORECAST, $L = 100$	3857	40%	134%
FORECAST, $L = 200$	3899	41%	137%
FORECAST, $L = 400$	3821	40%	132%
FORECAST, $L = 800$	3825	40%	133%
FORECAST, $L = 1600$	3732	39%	127%
Optimum	5271	55%	220%

We can see that FORECAST clearly outperforms the random selection strategy. To provide concrete numbers, we have picked 15% of the simulated sandbox capacity to compare the approaches, because the more limited the resources are, the more important it is which samples are selected for analysis. The results are shown in Table 4. With 15% sandbox capacity and $L = 100$, FORECAST allows us to observe 134% more potential C&C servers compared to a random selection strategy. Furthermore, both peHash and static perform significantly better than random selection, by 52% and 36% respectively. This confirms the intuition from previous work [41, 23] that diversity is a good heuristic for sample selection. Nonetheless, FORECAST provides noticeable benefits compared to both strategies, outperforming peHash by 54%, and static by 72%. This demonstrates that explicitly optimizing sample selection for an analysis goal can improve the overall value of the analysis results. It is also worth noting that the optimum selection strategy outperforms FORECAST by 37%. This shows that there is some margin for improving FORECAST's performance if the accuracy of its cluster prediction component can be increased, for instance by considering additional static features or by improving the classifier. Table 4 also shows that higher levels of parallelism have modest negative effects on performance. With $L = 1600$, FORECAST reveals only 4% less features than with $L = 50$. Note that a parallelism level of 1600 is over an order of magnitude larger than our current deployment.

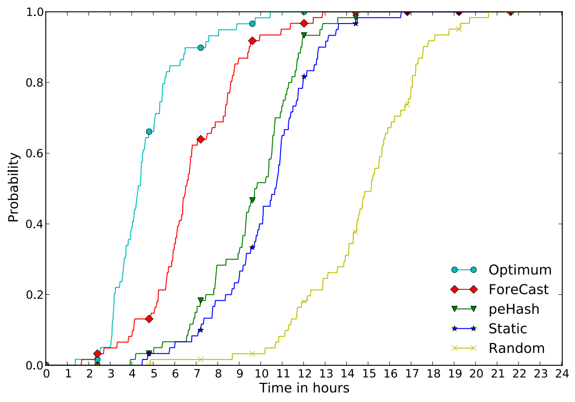


Figure 4: CDF of necessary samples to get 60% of network features, $L = 100$

Figure 4 shows the cumulative distribution function, over the 61 days considered, of the time needed by a simulated sandbox with 100% capacity to observe 60% of relevant features. We can see that on the median day, a sandbox using FORECAST could observe 60% of features after having analyzed only about one third of the daily samples in 7.2 hours. A sandbox using the random selection strategy, on the other hand, would need 15 hours to provide the same number of potential C&C servers, while with the diversity-based sample selection strategies, about 10 hours would be needed. A secondary benefit of FORECAST is therefore the faster response to new C&C servers.

4.4 Performance

Table 5: FORECAST **run-time on 2010 dataset,** $L = 100$

	Total (hours)	Per Sample (s)
Feature Extraction	48	0.39
Cluster Prediction	13	0.11
Cluster Scoring	0.34	<0.01
Clustering	17	0.13
Total	78.34	0.64

Since the aim of a sample selection strategy is to more efficiently use the computing resources available for dynamic analysis, it cannot itself require excessive resources. Clearly, deciding if a sample should be analyzed must be much faster than actually running dynamic analysis on it.

Table 5 shows FORECAST's run-time for the simulation described in the previous section on the 2010 dataset, running on a single server. As we can see, the total time per sample is under one second. This is negligible compared to the four minutes our sandbox spends executing each sample. Note that the cost of running AV engines on the samples is not included in this figure, because we obtain AV results from the VirusTotal service [7]. Performing AV-scanning with all engines supported by VirusTotal would require an additional five seconds per sample using a single machine [16].

4.5 Evasion

Our experiments have shown that FORECAST is effective in selecting for dynamic analysis samples that will provide useful information. However, malware authors could attempt to avoid analysis by tricking our system into *not* selecting their binaries. For this, they could attack our cluster prediction component, which ultimately relies on the static features discussed in Section 3.2. A first approach would be to mutate malware samples so that our system cannot statically recognize variants as similar. For this, malware authors could develop techniques for polymorphic mutation designed to evade peHash [41] as well as static clustering [23]. Furthermore, they could try to confuse the AV companies' (proprietary) techniques for assigning names to malware variants. If such mutation techniques were successful and widespread, FORECAST would become at best useless. However, an individual malware author has little incentive to deploy such a technique, because it would not succeed in evading analysis for his samples. The reason is that our cluster prediction component would most likely assign such novel-looking samples to the other cluster O. As we can see from Table 3, because of its diversity, the other cluster is ranked quite high by our cluster scoring algorithm.

Alternatively, a malware author could attempt to perform a mimicry attack, tricking FORECAST into assigning his malware to a cluster that has very little interesting behavior (such as the Allaple cluster shown in Table 3). Indeed it is relatively straightforward to develop a sample that resembles a variant of the Allaple worm. However, such a sample would hardly be successful, as it would be immediately detected by most AV engines. Evading detection by AV engines while performing mimicry against FORECAST's cluster prediction (which includes AV labels among its features) would seem to be challenging.

5. RELATED WORK

There exists a large body of related work on malware detection and analysis. Currently, the most popular approach for malware analysis relies on sandboxes [13, 3, 4, 5, 6]. A sandbox is an instrumented execution environment that runs an unknown program, recording its interactions with the operating system (via system calls) or other hosts (via the network). Often, this execution environment is realized as a system emulator or a virtual machine. For each malware program that is analyzed, a sandbox will produce a report that details the host-based actions of the sample and the network traffic that it produces.

Based on the reports that capture the dynamic activity of malware programs, it is possible to find clusters of samples that perform similar actions [11, 14], or to perform supervised malware classification to detect samples from known malware families [36].

In addition to approaches that perform malware clustering and classification on the output of sandboxes, there are a number of static techniques that share the same goal but operate directly on the malware executable [26, 35, 25, 40]. Unfortunately, these tools all assume that the malicious code is first unpacked and disassembled. However, existing generic unpackers rely on the dynamic instrumentation of executables [37, 28, 24]. That is, these systems need to execute the sample. This is a problem in our context, because we aim to avoid the overhead associated with dynamic analysis and need to pick a sample without executing the malware first.

A few tools can process packed malware samples but do not require a previous, dynamic unpacking step. Some of these tools [33] do not attempt to classify (or cluster) malware programs directly. Instead, they use static analysis only to distinguish between packed and unpacked executables. Packed executables are then forwarded to a dynamic unpacker. We are only aware of two systems that can detect duplicate malware samples using a fully static approach [41, 23]. Our system uses both of these tools to produce input that we then leverage for the cluster prediction step. As our experiments demonstrate, FORECAST significantly outperforms these systems.

Finally, [12] takes a dynamic approach to duplicate sample detection: All samples are executed in the sandbox, but after a short time-out (1 minute) the behavior so far is compared with the behavior of previously analyzed samples. If the sample is detected as a duplicate, execution is immediately terminated, otherwise analysis continues until a longer analysis time-out. The authors do not evaluate their approach with respect to an objective function. In any case, their system requires at least one minute to discard "uninteresting" samples, while FORECAST spends only 0.64 seconds on each sample, as shown in Table 5.

6. CONCLUSION

Given the flood of tens of thousands of malware samples that are discovered every day, time is a valuable resource for a dynamic malware analysis system. Of course, the available time should be spent on analyzing samples that are most relevant, where relevance depends on the goals of the analyst and the application domain. For example, for botnet C&C analysis, one would prefer to pick samples that produce network traffic and reveal the locations of C&C servers.

In this paper, we presented FORECAST, a system that can select the malware sample that is most likely to yield relevant information, given a domain-specific scoring function. The key requirement is that this selection process has to be performed efficiently, on a possibly large pool of candidates, and without actually running a sample. To realize our approach, we use a large number of input features that are statically extracted from malware executables. These features are then fed into a predictor, which estimates the expected information gain when executing a sample. This predictor uses machine learning techniques and leverages a knowledge base built from previously-analyzed samples. Our experiments demonstrate that FORECAST is effective in selecting interesting samples. On a test set of more than 600 thousand malware samples, our system showed a high accuracy and significantly outperformed a selection strategy that simply avoids picking similar (or duplicate) samples.

7. ACKNOWLEDGEMENTS

The research leading to these results has received funding from the European Union Seventh Framework Programme under grant agreement n. 257007 (SysSec), from the Prevention, Preparedness and Consequence Management of Terrorism and other Security-related Risks Programme European Commission - Directorate-General Home Affairs (project i-Code), and from the Austrian Research Promotion Agency (FFG) under grant 820854 (TRUDIE).

8. REFERENCES

[1] Adblock List. Retrieved September 2010 from `https://secure.fanboy.co.nz/fanboy-adblock.txt`.

[2] Anubis. `http://anubis.iseclab.org`.

[3] CWSandbox. `http://www.cwsandbox.org`.

[4] Joebox: A Secure Sandbox Application for Windows. `http://www.joebox.org/`.

[5] Norman Sandbox. `http://sandbox.norman.com`.

[6] ThreatExpert. `http://www.threatexpert.com`.

[7] Virustotal. `http://www.virustotal.com`.

[8] Zeus Tracker. `https://zeustracker.abuse.ch`.

[9] Panda Labs Annual Report, 2010.

[10] E. L. Allwein, R. E. Schapire, and Y. Singer. Reducing Multiclass to Binary: A Unifying Approach for Margin Classifiers. In *International Conference on Machine Learning*, 2000.

[11] M. Bailey, J. Oberheide, J. Andersen, Z. Mao, F. Jahanian, and J. Nazario. Automated Classification and Analysis of Internet Malware. In *Symposium on Recent Advances in Intrusion Detection (RAID)*, 2007.

[12] U. Bayer, E. Kirda, and C. Kruegel. Improving the Efficiency of Dynamic Malware Analysis. In *Symposium On Applied Computing (SAC)*, 2010.

[13] U. Bayer, C. Kruegel, and E. Kirda. TTAnalyze: A Tool for Analyzing Malware. In *Annual Conf. of the European Institute for Computer Antivirus Research (EICAR)*, 2006.

[14] U. Bayer, P. Milani Comparetti, C. Kruegel, and E. Kirda. Scalable, Behavior-Based Malware Clustering. In *Network and Distributed System Security (NDSS)*, 2009.

[15] H. Binsalleeh, T. Ormerod, A. Boukhtouta, P. Sinha, A. Youssef, M. Debbabi, and L. Wang. On the analysis of the zeus botnet crimeware toolkit. In *International Conference on Privacy, Security and Trust*, 2010.

[16] J. Canto. VirusTotal Timing Information. Hispasec Systemas. Private communication, 2011.

[17] A. Decker, D. Sancho, L. Kharouni, M. Goncharov, and R. McArdle. A study of the Pushdo / Cutwail Botnet. `http://us.trendmicro.com/imperia/md/content/us/pdf/threats/securitylibrary/Study_of_pushdo.pdf`, 2009.

[18] M. Dredze and K. Crammer. Confidence-Weighted Linear Classification. In *International conference on Machine learning*, 2008.

[19] R.-E. Fan, K.-W. Chang, C.-J. Hsieh, X.-R. Wang, and C.-J. Lin. LIBLINEAR: A Library for Large Linear Classification. *Journal of Machine Learning Research*, 9, 2008.

[20] T. Hastie and R. Tibshirani. Classification by Pairwise Coupling. In *Advances in neural information processing systems*, 1997.

[21] T.-K. Huang, R. C. Weng, and C.-J. Lin. Generalized Bradley-Terry Models and Multi-Class Probability Estimates. *Journal of Machine Learning Research*, 7, 2006.

[22] G. Jacob, R. Hund, T. Holz, and C. Kruegel. JACKSTRAWS: Picking Command and Control Connections from Bot Traffic. In *USENIX Security Symposium*, 2011.

[23] G. Jacob, M. Neugschwandtner, P. Milani Comparetti, C. Kruegel, and G. Vigna. A static, packer-agnostic filter to detect similar malware samples. Technical Report 2010-26, UC Santa Barbara, 2010.

[24] M. G. Kang, P. Poosankam, and H. Yin. Renovo: A Hidden Code Extractor for Packed Executables. In *ACM Workshop on Recurring malcode (WORM)*, 2007.

[25] A. Karnik, S. Goswami, and R. Guha. Detecting Obfuscated Viruses Using Cosine Similarity Analysis. In *Asia International Conference on Modelling & Simulation*, 2007.

[26] J. Kolter and M. Maloof. Learning to Detect Malicious Executables in the Wild. *Journal of Machine Learning Research*, 7, 2006.

[27] B. Krebs. Takedowns: The Shuns and Stuns That Take the Fight to the Enemy. *McAfee Security Journal*, (6), 2010.

[28] L. Martignoni, M. Christodorescu, and S. Jha. Omniunpack: Fast, generic, and safe unpacking of malware. In *Annual Computer Security Applications Conf. (ACSAC)*, 2007.

[29] A. Mushtaq. Smashing the Mega-d/Ozdok botnet in 24 hours. `http://blog.fireeye.com/research/2009/11/smashing-the-ozdok.html`, 2009.

[30] M. Neugschwandtner, P. Milani Comparetti, G. Jacob, and C. Kruegel. FORECAST - Skimming off the Malware Cream (extended version). Technical Report TR-iSecLab-0911-001, Vienna University of Technology, 2011.

[31] R. Paleari, L. Martignoni, E. Passerini, D. Davidson, M. Fredrikson, J. Giffin, and S. Jha. Automatic Generation of Remediation Procedures for Malware Infections. In *USENIX Security Symposium*, 2010.

[32] V. Paxson. Bro: A System for Detecting Network Intruders in Real-Time. In *USENIX Security Symposium*, 1998.

[33] R. Perdisci, A. Lanzi, and W. Lee. McBoost: Boosting Scalability in Malware Collection and Analysis Using Statistical Classification of Executables. In *Annual Computer Security Applications Conference (ACSAC)*, 2008.

[34] R. Perdisci, W. Lee, and N. Feamster. Behavioral Clustering of HTTP-Based Malware and Signature Generation Using Malicious Network Traces. In *USENIX conference on Networked Systems Design and Implementation*, 2010.

[35] K. Reddy, S. Dash, and A. Pujari. New Malicious Code Detection Using Variable Length n-grams. In *Information Systems Security Conference*, 2006.

[36] K. Rieck, T. Holz, C. Willems, P. Düssel, and P. Laskov. Learning and Classification of Malware Behavior. In *Detection of Intrusions and Malware and Vulnerability Assessment (DIMVA)*, 2008.

[37] P. Royal, M. Halpin, D. Dagon, R. Edmonds, and W. Lee. PolyUnpack: Automating the Hidden-Code Extraction of Unpack-Executing Malware. In *22nd Annual Computer Security Applications Conf. (ACSAC)*, 2006.

[38] D. Sancho. You Scratch My Back... Bredolab's Sudden Rise in Prominence. `http://us.trendmicro.com/imperia/md/content/us/trendwatch/researchandanalysis/bredolab_final.pdf`, 2009.

[39] B. Stone-Gross, A. Moser, C. Kruegel, K. Almaroth, and E. Kirda. FIRE: FInding Rogue nEtworks. In *Annual Computer Security Applications Conference (ACSAC)*, 2009.

[40] S. Tabish, M. Shafiq, and M. Farooq. Malware Detection Using Statistical Analysis of Byte-Level File Content. In *ACM SIGKDD Workshop on CyberSecurity and Intelligence Informatics*, 2009.

[41] G. Wicherski. peHash: A Novel Approach to Fast Malware Clustering. In *2nd USENIX Workshop on Large-Scale Exploits and Emergent Threats (LEET)*, 2009.

Detecting Malware's Failover C&C Strategies with SQUEEZE

Matthias Neugschwandtner[1], Paolo Milani Comparetti[1], and Christian Platzer[1]

[1]Vienna University of Technology, {mneug,pmilani,cplatzer}@seclab.tuwien.ac.at

ABSTRACT

The ability to remote-control infected PCs is a fundamental component of modern malware campaigns. At the same time, the command and control (C&C) infrastructure that provides this capability is an attractive target for mitigation. In recent years, more or less successful takedown operations have been conducted against botnets employing both client-server and peer-to-peer C&C architectures. To improve their robustness against such disruptions of their illegal business, botnet operators routinely deploy redundant C&C infrastructure and implement failover C&C strategies.

In this paper, we propose techniques based on multi-path exploration [1] to discover how malware behaves when faced with the simulated take-down of some of the network endpoints it communicates with. We implement these techniques in a tool called SQUEEZE, and show that it allows us to detect backup C&C servers, increasing the coverage of an automatically generated C&C blacklist by 19.7%, and can trigger domain generation algorithms that malware implements for disaster-recovery.

1 Introduction

Malicious code, also known as malware, is an essential component of criminal activity on the internet. Miscreants use a variety of strategies to infect computers with malware and organize them into networks of remote-controlled *bots*. These *botnets* can then be used for a variety of harmful activities. These include identity theft (such as stealing a user's credit card number or online banking credentials), sending out unwanted email (SPAM), performing distributed denial of service attacks (DDoS), tricking the user into purchasing fake anti-virus products, or generating advertisement revenue by producing fake "clicks" on advertisement links. In fact, internet criminals are always looking for new ways to profit from the computers they control at the expense of their legitimate users or of the internet at large.

To be successful and maximize their profits, botnet operators need to be able to dynamically control and update their malware installations. This allows them to adapt a botnet's behavior to the dynamic, adversarial environment in which they operate. For instance, a successful SPAM operation has to frequently modify the structure and content of the messages being sent, not only because the goals of the campaign may change rapidly over time (from advertising online pharmacy web-sites, to distributing attachments that contain an exploit) but also because they need to remain one step ahead of anti-SPAM efforts.

The Command and Control (C&C) infrastructure that botmasters use to control and update their bots is thus a critical component of their operations. At the same time, C&C is an attractive target for those who wish to mitigate the damage caused by malware. Disabling a botnet's C&C infrastructure can effectively *take down* the botnet. If botmasters lose control of their bots, they are prevented from using them to cause further mischief, even though the malware infections of the individual bots may not themselves have been remediated.

In recent years, botnet operators have therefore deployed a variety of client-server or peer-to-peer (P2P) C&C architectures, designed to be robust against take-down attempts. Recent client-server botnets include Pushdo/Cutwail [2], Torpig [3], Rustock [4] and Zeus (strictly speaking, Zeus is a toolkit for building botnets [5]). Examples of P2P botnets include Storm [6, 7], Nugache [8] and Waledac [9]. A few botnets, such as later versions of Conficker, combine both C&C approaches [10]. Modern client-server botnets do not naively rely on a single C&C server. Instead, they try to achieve robustness using domain flux [3], frequent updates of the C&C endpoints, and a high level of redundancy. For instance, the Koobface botnet uses about one hundred C&C servers running on compromised hosts [11]. In many cases, the botnets are also partitioned such that a single bot installation does not contain the coordinates of the entire C&C infrastructure.

Nonetheless, both client-server and P2P botnets have been the target of more or less successful node enumeration [12], infiltration [7, 3, 13, 11, 14] or take-down [15, 16, 17, 18] operations. Most recently Rustock, one of the largest SPAM botnets, was successfully taken down as a result of legal action led by Microsoft [19]. From a technical point of view, this involved identifying all of the C&C servers employed by the botnet, and taking them down simultaneously. Furthermore, the botmasters had to be prevented from registering specific domains in China that they could have used to recover control of their bots. In some cases it may not be feasible to take down identified C&C servers in a timely manner. However, even in such cases a blacklist of C&C servers, such as the ones provided by FIRE [20] or Zeus Tracker[1], can be extremely useful. By deploying such a blacklist, network administrators are able to detect infected machines in their network and to prevent them from receiving further commands from the botmasters.

A common way of building a C&C blacklist is to run malware samples in an analysis sandbox such as Anubis [21] or CWSandbox [22] and detect the C&C traffic they generate [20, 23, 24, 25, 26]. This approach, however, suffers from the familiar limitations of dynamic analysis: incomplete coverage. That is, a single execution of a malware sample is unlikely to reveal several, let alone all, of the redundant C&C servers that the bot is able to contact. Furthermore, so long as the botnet's main C&C servers are available,

[1]`https://zeustracker.abuse.ch/`

the bot will not reveal any fallback strategies it may be provided with to recover from C&C takedown. For instance, after all of its C&C servers were taken down in 2008, the Srzbi botnet was able to get back online because it implemented a domain generation algorithm (DGA) as a recovery mechanism [16].

The goal of this work is to improve the amount of C&C behavior that can be observed during malware execution. Specifically, our aim is to increase the number of C&C servers that are contacted during an analysis run and can therefore be detected, and to trick malware into revealing backup C&C strategies such as DGA algorithms. For this, we use a specialized, targeted form of multi-path exploration [1]. The basic idea is that, if a bot is blocked from contacting a specific C&C server, it may try to contact alternative C&C endpoints.

When a bot attempts to contact an endpoint, an analysis system can either allow the communication to proceed, or block it. By making such a decision for each endpoint, we are essentially exploring a binary tree of execution paths. This tree grows in size exponentially with the number of endpoints that a sample may contact. During its execution, a malware sample frequently contacts a significant number of endpoints, most of which are not related to C&C, and are, therefore, uninteresting for our purposes. Therefore, it is not feasible in practice to explore the entire execution tree. Furthermore, before C&C behavior can be observed, we may need to allow the bot to contact specific endpoints, that it uses to test the state of its internet connection or to download additional code. Therefore, intelligent strategies are needed to explore the execution tree and discover the largest amount of C&C endpoints within a reasonable time frame. In Section 3, we introduce two such strategies. Furthermore, fast exploration of the execution tree requires the ability to revert the state of the analysis environment to the moment before a connection was blocked or allowed, to immediately explore the alternative branch.

In this work, we introduce a system called SQUEEZE, that is able to increase the amount of C&C endpoints and strategies that are revealed during dynamic analysis. We evaluate our tool on over 8000 malware samples. Results show that, even with a relatively short run-time, SQUEEZE can reveal hundreds of domains and IPs of C&C servers that were never observed in a normal run of our Anubis sandbox. As a result, we are able to increase the number of entries in an automatically-generated C&C blacklist by 19.7%. Furthermore, we show that our tool can reveal malware's ability to use failover strategies such as DGA algorithms.

In summary, our contributions are the following:

- We introduce two effective strategies for exploring the tree of executions generated by a malware sample when faced with the availability or absence of contacted network endpoints.
- We design and implement a system that is able to efficiently execute multiple paths within this tree. For this, we employ techniques for reverting the execution state of the analysis sandbox and dynamically re-configuring its network environment.
- We evaluate the proposed techniques on a diverse and representative collection of real-world malware samples, and show that they lead to the detection of hundreds of additional C&C endpoints. This allows us to increase the size of an automatically-generated C&C blacklist by 19.7%. Furthermore, we show that SQUEEZE can reveal alternative C&C strategies that are used by malware to recover from the takedown of its primary C&C infrastructure.

2 Approach

To discover backup C&C servers and fallback strategies, we use a dynamic approach. That is, we run malware samples in a controlled environment and observe their C&C communication. Our approach is targeted against client-server C&C architectures. The basic idea is to simulate the takedown of a malware's primary C&C servers to trick it into revealing the servers and algorithms it is provided with for failover. For this, the obvious approach would be to simply block all traffic originating from the sandbox environment. There are, however, several reasons why this does not work in practice. First of all, a bot binary may be delivered by a dropper, as is the case in pay-per-install affiliate programs. In such cases, a naive approach would not allow the interesting bot binary to be downloaded in the first place. Furthermore, malware authors frequently contact a popular server to check their internet connectivity. If it is not available, the sample quits or idles until the network becomes available. Simply dropping all traffic would cause exactly this undesired behavior. Finally, unless we allow traffic to potential C&C endpoints, we are unable to verify if the traffic is indeed related to C&C activity.

These considerations provide us with a starting point for the design of a system for triggering malware's failover C&C. First of all, we need to treat C&C traffic differently than other network traffic. Traffic that is not related to C&C communication should be allowed through, to the extent that this is possible without allowing the malware to cause harm. On the other hand, C&C traffic should be blocked to trigger backup behavior. For this, we require some knowledge about C&C communication, such as signatures for C&C traffic, that can be used to detect new C&C endpoints. Finally, to trick the malware into communicating with several of its redundant C&C servers, we need to be able to block C&C traffic *after* having allowed through enough of it for our models of C&C communication to detect it. For this, we need to "rewind" the malware execution to the moment before the C&C connection was successfully established. This can be achieved by reverting to a previously taken snapshot in a virtual machine.

Our approach is essentially a specialized form of multi-path exploration [1]. For this, it relies on a knowledge base on malware C&C communication, and on snapshotting functionality that allows it to explore multiple execution paths within a single analysis run. Whenever an analyzed sample attempts to contact a new endpoint, we take a snapshot before deciding whether to allow this traffic. Later in the analysis, we are able to revert to this snapshot to explore the alternative branch. The C&C knowledgebase provides some domain knowledge to help us decide which execution branches to explore. This knowledge could come in the form of network-based signatures for C&C traffic [24, 25] or of network-based [23] or host-based [26] behavioral models.

3 Exploration Strategies

To describe possible strategies for exploring a malware sample's network behavior on different execution paths, we will use a running example. Figure 1 shows the different components of a malware, the endpoints it connects to and the effect that connectivity to these endpoints has on its network behavior. In this example a dropper is used to distribute the malware. As a first step, the dropper will connect to a remote server (`File-Server A`) and download the actual bot component. If the server is not accessible, the dropper employs a simple failover strategy and will try to contact backup server `File-Server B`. If either download is successful, the downloaded payload is executed, launching the actual bot. In this case, the dropper does not actually save the payload to disk before running it. Instead, it injects the downloaded code into another process and starts a remote thread [2]. Before executing any malicious behavior, the bot performs a connectivity check by attempting to access a popular web site, in this case the Yahoo home page. If the connectivity check fails, no further action is taken. Otherwise, the bot registers itself at the primary C&C server and receives further instructions. These commands will cause the bot to launch a spam engine or a port scan, connecting to even more endpoints. The endpoints that are of primary interest to us, however, are those associated with C&C communication. If the primary C&C server is unreachable, our example tries to connect to a different hard-coded endpoint. Only if this backup is also unavailable, the bot falls back to a Domain Generation Algorithm. The DGA generates large numbers of domains based on the current date (obtained by parsing the Yahoo home page, rather than from the system clock). In case the two main C&C servers were taken offline, the botmasters would register a few of these domains each day, and use them to provide their bots with an updated list of C&C servers.

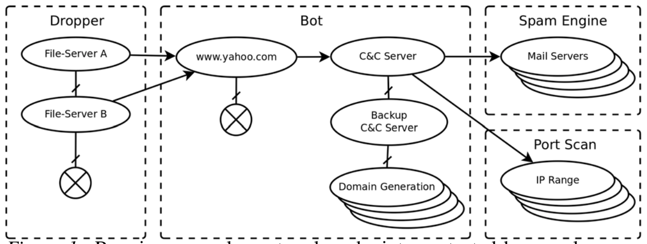


Figure 1: Running example: network endpoints contacted by a malware sample. A line between two endpoints indicates that the malware attempts to contact the endpoints one after the other. Lines with an arrow represent behavior following a successful connection, while crossed lines originate from unreachable endpoints. A crossed circle indicates no further network activity.

Every time SQUEEZE encounters an endpoint a decision has to be taken on whether to block this endpoint or allow the communication to succeed, influencing the behavior of the malware and the further endpoints it will or will not contact as a result. By representing the endpoints as nodes, the decision whether or not to block a connection as node expansion and the result as edges, the process of analyzing a malware's network behavior can be modeled as the exploration of a binary tree. The observant reader will notice that Figure 1 is in fact a directed acyclic graph, rather than a tree, since the node `www.yahoo.com` can be reached from two different paths. However, malware may exhibit different behavior depending on the sequence of endpoints it was able to contact. For instance, the two file servers in Figure 1 may in fact deliver different payloads. Strictly speaking, each node in the execution tree is therefore identified by the sequence of endpoints that the malware has attempted to contact so far, rather than by the latest endpoint only. Note that an endpoint may be identified by its domain name or by its IP address. We use the IP address only in cases where the IP was not previously obtained by the analyzed sample through DNS resolution.

Since we can analyze each malware sample for a limited amount of time, it is unlikely that we can explore all possible branches of the execution tree during an analysis run. In fact, some malicious behavior (most obviously, scanning) involves attempting to contact a practically unlimited number of endpoints. Therefore we need to develop a strategy for exploring the execution tree that will reveal the largest amount of C&C activities, given the time constraints. Standard approaches for traversing a tree include depth-first and breadth-first search algorithms. As we will see, neither approach is directly suitable to the problem at hand. Instead, we will need to develop exploration strategies that make use of domain knowledge on C&C communication.

In a breadth-first search, each node is completely expanded before proceeding to the next node. This would allow us to explore many branches of a sample's initial network behavior, but would not lead us very deep within the execution tree. Since C&C endpoints, tried one after the other by the malware, typically form a deep branch in the tree, depth is more important to us than breadth. A depth-first search would therefore seem better suited to our needs, but in fact it also has its pitfalls. With a depth-first search, exploration can easily get "stuck" in a branch where a lot of uninteresting network activity is occurring (such as port scanning or click fraud).

A solution to this problem is to enrich the search strategy with additional knowledge on C&C communication. Such knowledge can contribute to the decision whether a branch is of interest and its nodes should be expanded or not. Ideally, we would exactly know which nodes of the tree correspond to C&C endpoints. In practice, however, we may not be able to automatically identify all C&C communication. Nonetheless, since our goal is to automatically build a blacklist of C&C endpoints, it is necessary to have some means of identifying C&C traffic. For SQUEEZE, we use domain knowledge in the form of a set of network-based C&C signatures that were provided to us by a security company. These signatures have been vetted by human experts, so we have high confidence that they can identify C&C communication without generating false positives (though they may not cover all forms of C&C communication, leading to false negatives). Furthermore, by matching these signatures against the traffic observed during malware execution in our Anubis sandbox in the past, we can identify a set of known C&C endpoints. The C&C signatures and the set of known C&C endpoints constitute our initial knowledgebase on malware Command and Control.

There is a significant difference between these two types of information: A known C&C endpoint can be blocked *before* a connection is actually established, while a C&C signature can only be matched *after* traffic has already been transmitted. Hence, the former can be used with a block-first strategy while the latter requires an allow-first strategy. Another aspect to take into account is that each component of the malware – bot, spam engine, etc. – is typically executed in a separate process. Using our C&C knowledgebase, we are able to identify the process or processes that are carrying out C&C communication. We will call these processes the *bot component*. With this information, we can focus our efforts on exploring the parts of the execution tree that represent endpoints contacted by the bot component.

For SQUEEZE, we developed two alternative strategies for exploring the execution tree. Both can be seen as depth-first search algorithms, but they differ in the type of C&C knowledge that they leverage. Figure 2 provides an example for both approaches.

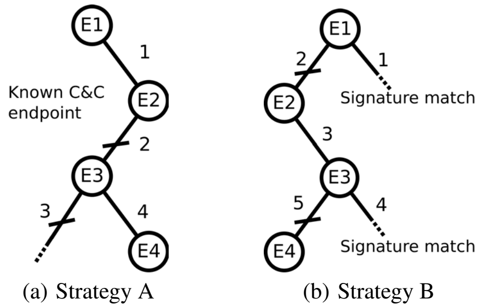


Figure 2: The two exploration strategies. A node represents an endpoint. A solid line represents an allowed connection, a crossed line a blocked connection and dotted lines backtracking before another endpoint is discovered.

Strategy A takes advantage of the set of known C&C endpoints. When using this strategy, SQUEEZE initially allows all connections. Once the analyzed malware attempts to contact a known

C&C endpoint, SQUEEZE blocks the connection, and switches to a block-first, depth-first search. After each decision to block or allow an endpoint, a timeout is restarted. If no further endpoints are contacted within the timeout, the search backtracks. Furthermore, whenever the tool detects a connection to a known C&C endpoint, it identifies the responsible process, and adds it to the bot component. The exploration strategy only takes into account endpoints contacted by the bot component, and always allows traffic to other endpoints. The idea behind this strategy is to allow the malware to perform connectivity checks or other network behavior that needs to be successfully completed before C&C communication is triggered. Once the malware attempts a first C&C connection, it is blocked, leading the bot to attempt to contact its backup C&C servers, one after the other. Note that this strategy will fail if no known C&C endpoints are observed.

Figure 2(a) shows an example of Strategy A in action: After allowing the connection to E1, the system recognizes the known C&C endpoint E2. Blocking E2 causes the malware to perform an intermediate connection check on endpoint E3. The connection is initially blocked, and the sample performs no further network behavior. After a timeout, we track back and unblock E3. Following the now successful connection check, the malware contacts its backup C&C server at endpoint E4.

Strategy B takes advantage of the set of C&C signatures. When using this strategy, SQUEEZE performs a depth-first, allow-first search. However, the C&C signatures are matched against all observed network traffic. As soon as a signature matches the traffic going to a specific endpoint, SQUEEZE backtracks, blocking communication with this C&C endpoint. This strategy is aimed at efficiently discovering endpoints that can be detected with the available C&C signatures.

Figure 2(b) shows an example of Strategy B: When a signature match is detected in traffic to E1, the system tracks back and blocks this endpoint. Since the following connection check on endpoint E2 is successful, the malware immediately proceeds to the backup C&C server E3. Traffic from E3 also matches a C&C signature. Blocking this endpoint reveals E4.

The two strategies have different advantages and disadvantages. By using a block-first approach, strategy A tends to reveal more endpoints overall. However, unless the exploration backtracks far enough to allow the malware to connect to an initially blocked endpoint, we do not observe traffic to it and are therefore unable to confirm if it is in fact a C&C server. Strategy B has the advantage that it will trigger on a C&C connection even if the contacted C&C endpoint is not known. Finally, an advantage of strategy A is that it is more flexible, and can leverage knowledge of C&C endpoints regardless of how they were detected. Strategy B, on the other hand, relies on the ability to detect C&C communication on the fly. It cannot be as easily combined with C&C detection approaches that rely on observing the behavior of a bot *after* it receives a command, such as those from Wurzinger et al. [23].

4 System Description

Figure 3 shows an overview of SQUEEZE's architecture. SQUEEZE takes as input a malware sample and a knowledgebase on C&C communication. Depending on the exploration strategy used, this knowledgebase can contain C&C network signatures or C&C server endpoints. While the sample is executing in the sandbox, its network activity is constantly monitored and compared with the C&C information. Whenever a new endpoint is encountered, we take a snapshot of the current state of the system. The exploration strategy then determines when to backtrack in the execution tree by reverting to a previous snapshot. Furthermore, the sandbox's network environment is dynamically modified by re-configuring a DNS recursor and a firewall. Finally, a delay analysis module within the sandbox tries to avoid delays between the malware's network connections by manipulating time (as observed from within the sandbox), to "fast-forward" malware execution.

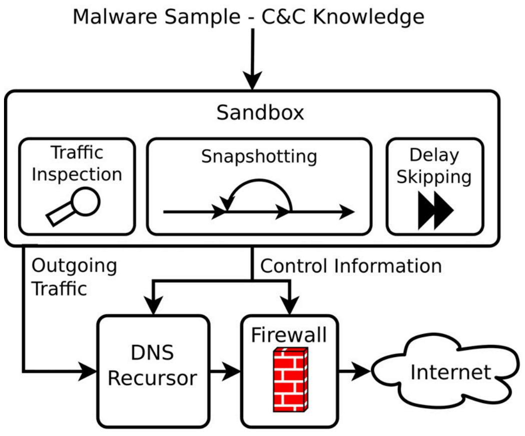


Figure 3: SQUEEZE Architecture

4.1 Sandbox

To build our analysis sandbox we leveraged our Anubis dynamic malware analysis system [21, 27]. Anubis is a dynamic malware analysis service that has been operating since early 2007, and has since analyzed over ten million malware samples. At its core is an instrumented full system emulator for the Windows XP operating system [21] and is built upon Qemu [28].

Snapshotting. To be able to backtrack in our exploration of the execution tree, we need to be able to revert the state not only of the malware being analyzed, but also of the operating system and analysis environment that it runs in. While it would in principle be possible to simply restart a sample's analysis with a different network configuration every time we need to backtrack, this would be extremely inefficient. Furthermore, in the new execution the malware might exhibit completely different behavior, confusing our exploration algorithms.

Since Anubis is a full system emulator, it is possible to take a snapshot of the state of the entire system (disk, memory and processor state), and to restore such a snapshot to revert the execution to the previous state. In fact, Qemu provides such a snapshotting mechanism. However, the Anubis instrumentation also keeps some execution state. For instance, it tracks which processes are under analysis and what resources operating system handles correspond to. To use Qemu snapshotting in Anubis, we therefore extend it to also save and restore the state of the Anubis instrumentation.

To reduce the time spent saving and restoring snapshots, we keep all generated snapshots in main memory, rather than save them to disk. Since we only have a limited amount of memory available, this imposes an upper bound on the number of snapshots that can be stored simultaneously. Since we use depth-first exploration strategies, this essentially corresponds to a limit to the depth of the execution tree that we can explore. With a suitable amount of memory this limit is seldom reached during SQUEEZE analysis.

Delay skipping. A general problem of dynamic analysis is that, for practical reasons, we are only able to execute each binary for a limited amount of time. For instance, in its default configuration, Anubis executes each sample for four minutes. As a consequence, malware could evade dynamic analysis by simply waiting a certain amount of time before performing any interesting behavior. For

SQUEEZE, this problem is exacerbated by the fact that malware frequently waits for some time before trying to contact a backup C&C server or resorting to a failover C&C strategy.

While this problem is hard to solve in general, in practice most malware samples can be tricked into skipping such delays. For this, we modify the behavior of system calls that can be used to delay a process' execution. As a first countermeasure, we intercept the undocumented `NtDelayExecution` native Windows API function, which returns after a specified time interval, and limit the delay to a maximum of half a second. Furthermore, we tamper with instructions and system calls, such as `GetTickCount`, that can serve as local sources of timing information, and could be used to busy-wait until a certain time has elapsed. Specifically, we send time forward by an interval that grows exponentially with the number of invocations of `GetTickCount`, up to a maximum value (currently set to five hours).

Traffic Interception. To be able to trigger the generation of snapshots and the dynamic re-configuration of the network environment required by our exploration strategy, SQUEEZE needs to match the network traffic generated by the malware against C&C endpoints and signatures on the fly. For this, we hook into the appropriate operating system interfaces and inspect their parameters *before* the actual functions are executed. Note that in Anubis, hooking happens from "outside the box" by instrumenting the emulator, rather than by modifying the operating system "inside the box". Specifically, we hook the `NtDeviceIoControlFile` function which is used to send data to device drivers. The driver used to transmit data over sockets is `afd.sys`, which is the Ancillary Function Driver for WinSock. Of the various I/O control codes used to handle connections, two are important for us: `AFD_CONNECT` which is used to establish a connection to a port at an IP address and `AFD_SEND` which sends input data out over an established link. To keep track of and be able to react to DNS queries, we additionally intercept the Windows DNS client's `DnsQuery` function that is implemented in `dnsapi.dll`.

4.2 Network Environment

SQUEEZE needs to dynamically adapt the network environment to block or allow connections to specific endpoints in accordance with the selected exploration strategy. For endpoints identified by a hostname, we want to block the hostname at the DNS level rather than the IP addresses associated with it. To this end we redirect all DNS queries to a local recursor. We use the popular PowerDNS recursor[2], because it allows fine-grained dynamic configuration using LUA scripts. This enables us to create an `NXDOMAIN` reply for DNS queries for hostnames we wish to block. To block IP addresses that are directly contacted by the malware (without a DNS lookup), we leverage the netfilter firewall[3] and configure it to reject the connection attempt and reply to a TCP SYN packet with a TCP reset, and to a UDP packet with an ICMP destination unreachable packet. This is more efficient than simply dropping the packet, because it avoids unnecessary delays caused by waiting for connection time-outs on the client side.

In addition to the dynamic network blocking required for SQUEEZE, we also need to prevent the analyzed samples from causing harm to the internet at large. For this, we deploy the same countermeasures employed by Anubis. These include preventing SPAM by redirecting SMTP connections to a local mail server (that does not actually send the mail), limiting a malware sample's contribution to denial of service attacks by throttling network throughput and number of connections, and blocking frequent attack vectors by redirecting traffic on a number of ports (such as TCP ports 139 and 445) to a local honeypot. These measures cannot completely guarantee that the malware will not cause any harm (0-day attacks in particular are hard to block). However, experience from the Anubis deployment shows that they are in practice sufficient for the safe operation of a malware analysis sandbox. Over four years running Anubis, we have received only a single abuse complaint, which was promptly addressed.

5 Evaluation

To evaluate SQUEEZE, we tested it on several thousand malware samples that had generated C&C traffic in Anubis. As we will show, analyzing these samples with SQUEEZE revealed a significant number of C&C servers that were not observed in Anubis.

To detect C&C traffic, we use a set of 192 network signatures provided to us by a security company. These signatures are similar in expressiveness to Snort rules [29], and include regular expressions that are to be matched against network flows. These signatures have been manually vetted for accuracy, and do not in our experience lead to false positives when matched against Anubis traffic. However, they may not cover the full spectrum of current malware C&C communication.

5.1 Dataset

By matching the C&C signatures against traffic dumps generated by Anubis analysis runs, we identify a set of samples that generated C&C traffic, together with the C&C endpoints they connected to. For our evaluation, we then selected a subset of these samples based on two criteria: First of all, we required recent samples, since older samples are often no longer functional because they do not contain up-to-date information for contacting C&C infrastructure. Second, we would like to test our tool on a dataset involving the largest possible set of C&C endpoints. Thus, we tried to contact each C&C endpoint and discarded those that were no longer reachable. This includes domains that failed to resolve to an IP address and servers that were no longer reachable on the port used for C&C. For each of the remaining endpoints, we selected the most recent samples that communicated with it.

In total, we selected 8,346 malware samples. To assess the diversity of this dataset, we scanned the samples with the Kaspersky anti-virus engine. Kaspersky provided a total of 2,225 different AV labels for these samples. Kaspersky virus labels have a loosely structured format, such as "`Trojan-Spy.Win32.Zbot.aebi`". By discarding the last part of these labels (indicating a specific variant), we can obtain a more coarse-grained classification of malware binaries into families. Our dataset contained 213 malware families according to this classification, with the most represented ten families accounting for only 21% of the dataset. Comparing these labels with a recent list of the most prevalent malware families [30] shows that fourteen of the twenty top malware families are represented in our dataset.[4] This confirms that we are testing our tool on a diverse and representative malware dataset. Notable exceptions (top 20 malware families not seen in our dataset) are Conficker and Storm 2.0, which are P2P botnets and are therefore not covered by our signatures for client-server C&C activity.

Setup. To run SQUEEZE we used four virtual machine instances on a host with 32GB memory and an Intel XEON E5420 CPU. 4GB of memory was assigned to each machine of which 3GB were dedicated to storing snapshots. Since snapshots are on average 130MB in size, this allows for approximately 21 snapshots to be taken. We run each malware sample for up to 6 minutes. Our setup can therefore analyze about one thousand samples a day. We ran our eval-

[2]http://www.powerdns.com

[3]http://www.netfilter.org

[4]A fifteenth family, the Bredolab botnet, is absent from our dataset because it was successfully taken down shortly after the report was published [31].

uation for 10 days in March 2011. Since our setup did not allow us to deploy the two strategies in parallel, the system was configured to use exploration strategy A during the first five days, and strategy B during the last five days. Our evaluation dataset is therefore split into Dataset A, consisting of 4,013 samples analyzed with SQUEEZE using strategy A, and Dataset B, consisting of 4,333 samples analyzed using strategy B. We used a new dataset for strategy B, rather than repeat analysis on Dataset A, to ensure we were testing SQUEEZE against a "fresh" selection of malware samples.

5.2 Malware Behavior

	Strategy A	Strategy B
Samples analyzed	4013	4333
Initial C&C knowledge match	54%	58%
No further activity	44%	43%
Substantial delay skipping	34%	31%
New endpoints	25%	23%
New endpoints in bot component	19%	13%
New endpoints with signature match	9%	8%

Table 1: Malware behavior in SQUEEZE

Table 1 shows an overview of the behavior of malware samples when analyzed with SQUEEZE using the two proposed exploration strategies. The first thing we notice is that a significant fraction of the analyzed samples never trigger SQUEEZE's blocking mechanisms. With strategy A, only 54% of analyzed samples ever try to contact a known C&C endpoint. Likewise, with strategy B only 58% of analyzed samples generate traffic that matches our C&C signatures. This is despite the fact that all of the samples in our dataset, when originally analyzed in Anubis, had performed signature-matching traffic that led us to detect an endpoint in the first place. We have found that in some cases the original binary is actually a downloader: In the Anubis execution, it dropped a remote-controlled bot, but in the SQUEEZE execution it dropped a completely different payload. In other cases some of the endpoints contacted by the sample were no longer available. One reason for this is that, during this experiment, the delay between executing a sample in Anubis and in SQUEEZE was too high (several days). Results from our deployment (Section 5.5) show that this result can be improved by reducing this delay.

A large share of the samples, 44% and 43% respectively for strategies A and B, do not employ a backup strategy at all. That is, after a C&C endpoint is blocked they show no further network activity. The difference between these first two rows of Table 1 leaves 10% and 15% of samples, respectively, where SQUEEZE may have been able to trigger additional C&C activity. As can be seen in the last row, 9% and 8% of samples respectively reveal new C&C endpoints when ran in SQUEEZE compared to those they revealed in Anubis. These endpoints are confirmed by a match against a C&C signature.

There may be several reasons why, for the remaining samples, no C&C traffic can be detected. First of all, with the block-first strategy of approach A, C&C traffic can only occur if the endpoint is unblocked before the end of the analysis. Then, backup C&C endpoints may not have been active during analysis. Finally, communication with a backup C&C server might use a slightly different format that cannot be detected by our C&C signatures. Therefore, 9% and 8% mark the lower bound of samples that revealed valuable endpoint information. A reasonable upper bound can be given by also including other endpoints that the bot component connected to. Recall that the bot component is defined as the set of processes that have performed C&C activity that matches our signatures or known endpoints. This provides an upper bound of 19% and 13% of samples respectively.

Furthermore, the results show that allowing connectivity checks to go through is important for SQUEEZE's effectiveness. Over ten percent of the samples in our dataset performed connectivity checks by contacting a variety of popular sites such as google.com, microsoft.com, or weather.yahoo.com. Some samples perform such checks only at startup, while others test their connectivity again whenever they fail to contact a C&C server.

Finally, our delay skipping techniques also play a significant role in SQUEEZE's effectiveness. For around one third of samples, SQUEEZE fast-forwards time in the sandbox by at least one minute. In Section 5.4 we will discuss an example of a malware family that reveals a significant number of C&C endpoints, but only after skipping a delay of over half an hour.

5.3 Endpoints

Table 2 shows the total number of distinct endpoints contacted when executing the samples in our datasets in an unmodified Anubis sandbox, as well as the additional endpoints contacted when using SQUEEZE with the two alternative exploration strategies.

The most important results are the numbers of C&C endpoints detected shown in the rows labeled "Endpoints with signature match". SQUEEZE is able to detect a significant number of C&C endpoints that were not contacted during the execution of any of the samples in datasets A and B. SQUEEZE reveals 201 and 185 new C&C endpoints respectively using strategies A and B. This corresponds to an increase in the number of C&C endpoints that can be extracted from our two malware datasets of 12.6% and 19.7% respectively. The number of IPs is significantly lower than the number of domains: Recall that we take into consideration an IP address only if it is accessed directly rather than obtained through DNS resolution. While some malware samples contain hard-coded C&C IP addresses, most rely on the DNS infrastructure to resolve the domain names of their backup C&C servers.

As discussed in the previous section, the number of endpoints with a signature match is only a lower bound for the amount of C&C endpoints we may have observed. This is chiefly because our C&C signatures may not match on all of a sample's C&C connections, particularly if the format of messages to the primary C&C server is not identical to the format used when communicating with a backup server. As a rough upper bound for C&C endpoints we can use all the endpoints contacted by the processes involved in C&C communication (the bot component). SQUEEZE may therefore have revealed up to 714 and 506 new C&C endpoints respectively using strategies A and B. This corresponds to an increase of 29.4% and 32.8% compared to unmodified Anubis.

To further investigate how many of the endpoints revealed by SQUEEZE are actually malicious, we used nine publicly available blacklists. For this, we queried these blacklists repeatedly until two months after running the malware samples. The results are shown in the row "Endpoints in blacklists". Table 3 shows a breakdown of these results for new endpoints (that is, all endpoints that were not contacted during the baseline Anubis analysis). Of the blacklists considered, only AMaDa focuses specifically on malware C&C endpoints. Several of the other blacklists include C&C endpoints as well as other malicious servers, while Google Safe Browsing and Norton Safe Web focus on malicious web sites. We nonetheless include these blacklists because miscreants may use endpoints for multiple malicious purposes. Note that all of these blacklists are meant to be deployed to block traffic to or from malicious hosts, and therefore strive to have extremely low false positive rates. The

Dataset A	**Baseline**			**Strategy A**		
	Domains	**IPs**	**Total**	**Domains**	**IPs**	**Total**
Endpoints	3562	767	4329	661	942	1603
Endpoints in bot component	2080	362	2432	454	260	714
Endpoints in blacklists	1970	111	2081	391	36	427
Endpoints with signature match	1489	110	1599	195	6	201
Dataset B	**Baseline**			**Strategy B**		
	Domains	**IPs**	**Total**	**Domains**	**IPs**	**Total**
Endpoints	2627	364	2991	534	325	859
Endpoints in bot component	1330	211	1541	293	213	506
Endpoints in blacklists	1336	81	1417	353	15	368
Endpoints with signature match	885	53	938	184	1	185

Table 2: New endpoints revealed by SQUEEZE compared to baseline (execution in Anubis sandbox)

Blacklist	**Strategy A**			**Strategy B**		
	Domains	**IPs**	**Total**	**Domains**	**IPs**	**Total**
Google Safe Browsing (Malware)	95	11	106	53	5	58
Norton Safe Web	47	N/A	47	80	N/A	80
Spamhaus (SBL,XBL,DBL)	2	20	22	2	7	9
ClearCloud DNS	281	N/A	281	226	N/A	226
DNS-BH Malware Domain Blocklist	101	N/A	101	86	N/A	86
Malware Domain List (MDL)	11	5	16	15	0	15
malc0de DNS Blackhole	80	3	83	70	1	71
Emerging Threats fwip rules	N/A	6	6	N/A	3	3
abuse.ch Malware Database (AMaDa)	32	2	34	40	2	42
Total Blacklisted	391	36	427	353	15	368
Total Endpoints	661	942	1603	534	325	859

Table 3: New endpoints revealed by SQUEEZE listed in publicly available blacklists

results are very different for domains and IP addresses: 62.5% of domains are listed in at least one blacklist, while the same is the case for only 4% of IPs. Overall, however, SQUEEZE has contacted 427 and 368 additional blacklisted endpoints using strategies A and B. Compared to the blacklisted endpoints revealed by Anubis, this is an increase of 20.5% and 26.0% respectively.

In conclusion, an estimate of the percentage of additional C&C servers revealed by SQUEEZE ranges between a minimum of 12.6% (confirmed C&C endpoints for strategy A) and a maximum of 32.8% (endpoints in bot component for strategy B). For automatically-generated C&C blacklists, such as the one provided by FIRE [20], these results have practical implications. The reason is that they demonstrate that, by complementing a malware analysis system with SQUEEZE, we can generate blacklists that provide a significantly improved coverage of malware C&C servers.

5.4 Qualitative Results

In this section, we will highlight some interesting aspects of the behavior of samples analyzed with SQUEEZE, with particular emphasis on those belonging to the top twenty malware families from the FireEye report [30].

Palevo/Butterfly is a botnet toolkit that, according to [30], is behind the currently most prevalent malware family. With SQUEEZE, we were able to observe the backup C&C strategies employed by Palevo samples. For this, strategy B was particularly effective. These samples initially attempt to contact a static IP address. This traffic matches our C&C signatures. After this endpoint is blocked, they attempt to contact another static IP (once again matching our signatures). Finally, they fall back to a domain generation algorithm. One sample in particular tried to resolve 42 generated domains during the analysis run, none of which were actually active.

Samples of the Pakes malware family fall back to a number of backup C&C endpoints when their primary C&C is blocked. Before attempting to contact each successive endpoint, however, these samples idle for several minutes. This delayed execution behavior is implemented with repeated calls to GetTickCount. Therefore, SQUEEZE was able to skip these delays. For one sample, analysis with strategy A revealed 20 additional endpoints compared to the original Anubis run, but only after sending time forward by over half an hour. As soon as Pakes is able to contact a C&C server, it receives instructions to send spam.

Koobface is another interesting malware family, because it straddles the line between client-server and P2P botnet C&C. As reported by Thomas et al. [11], Koobface sets up HTTP-based C&C servers on machines it has exploited. According to Thomas et al., close to one hundred such parasitic C&C servers are active on a given day. During analysis, Koobface samples initially performed a connection check by contacting the Google main page. Then they tried to contact a number of hardcoded C&C servers. By blocking them one after the other, SQUEEZE was able to trigger connections to up to fifty C&C endpoints before the six minute timeout. In this case, simply increasing the timeout would presumably allow us to detect further C&C endpoints.

The Piptea malware family (also known as Harnig) is a trojan downloader. During normal execution, samples contact the primary C&C server and query it via HTTP to retrieve encrypted download instructions. If the primary C&C server is unreachable, a second one serves as a fallback. However, the Piptea botnet was put offline, presumably by its operators, shortly after the Rustock takedown on March 17th, 2011 [32]. Since this was the first day of our evaluation period, we are able to observe this in our results. A few Piptea samples analyzed on March 17th successfully contacted their C&C servers. Those analyzed on a later date, however, could reach neither their primary nor their backup C&C servers.

Another interesting sample was also a downloader, most likely related to a pay-per-install affiliate program. This dropper queries a C&C server using a proprietary, plain-text protocol for which we do not have a signature. The server then provides a list of URLs from which to download additional executables. In our analysis, the dropper downloaded and executed three additional binaries one after the other. All three binaries are recognized by anti-virus engines as malicious, but interestingly do not seem to be related to each other: The dropper is thus installing multiple malware strains on the same machine. The third executable is a bot, and our signatures were able to detect its C&C traffic and the corresponding endpoint. When strategy B blocked this endpoint, the bot attempted to contact a backup C&C domain that had not yet been registered.

While SQUEEZE was able to reveal the backup C&C strategies employed by some malware families, there are also families that displayed no such behavior: After a C&C server was blocked, these samples simply idled or performed a default behavior such as network- or file-based propagation. We further investigated two such families, and concluded that indeed these malware samples do not employ a backup C&C strategy.

The first such family we investigated is Virut. Virut spreads by file infection, and its bot component is rather basic: The samples in our dataset use only a single IRC C&C server, which they contact to get download information on further binaries. If this server is unavailable, Virut's activity is limited to spreading.

The second family is Zbot. Zbot samples stem from the Zeus toolkit [5], which allows users to create their own, customized botnet. None of the samples we analyzed with SQUEEZE displayed a backup strategy when their first C&C server was blocked; In fact they remained completely idle. This seems surprising, considering that Zeus is still widely successful despite efforts from projects such as Zeustracker to identify and blacklist its C&C servers. The recently published source code of the Zeus toolkit (Version 2.0.8.9), however, allowed us to confirm this observation: Although zBot can use various C&C servers, it first relies on a primary C&C server to provide the bot configuration. This server is compiled into the binary and can thus only be changed by means of a binary update. To increase reslience, operators of Zeus-based botnets presumably rely on frequent updates to their bot binaries. If this is the case, a malware analysis system could reveal more Zbot C&C servers if it were to capture updated binaries downloaded during analysis, and periodically re-analyze the latest version. We leave this for future work.

5.5 Deployment

Based on the encouraging evaluation results we set up a stable deployment of SQUEEZE, integrated with the Anubis infrastructure. This deployment uses strategy B, and is configured to re-analyze all samples that matched a C&C signature in the original Anubis run. From the beginning of June until the end of August 2011, over 32.000 samples were re-analyzed by SQUEEZE. During this period, plain Anubis found a total of 4355 distinct endpoints in the bot component and 2338 distinct endpoints with a signature match. Squeeze improved this by revealing an additional 1706 (39%) and 278 (12%) endpoints respectively.

We tested the statistical significance of these results using a single-tailed Wilcoxon signed-rank test [33]. The research hypothesis states that an analysis with SQUEEZE reveals more endpoints than a plain Anubis run. The null hypothesis is rejected with a probability of error below 0.001% when applying the test to either endpoints in the bot component or endpoints with a signature match.

The deployment also backed our assumption that decreasing the delay until re-analysis would have a positive effect on the share of samples that re-connect to a C&C endpoint. While in Table 1 only 58% of samples re-analyzed with SQUEEZE matched a C&C signature, this number rises to over 90% in our live deployment, where the average delay between analysis in Anubis and re-analysis in SQUEEZE is reduced to eleven hours.

As a consequence, the FIRE service[5], which makes use of Anubis as a source of C&C information, directly benefits from SQUEEZE. This improves the coverage of both the FIRE blacklist and of its ranking of "malicious networks".

6 Limitations and Future Work

In this section we briefly discuss the main limitations of our approach. The first challenge is posed by the communication method used by botnets. First of all, our approach is of limited utility against botnets that use a fully decentralized P2P architecture. While SQUEEZE could still investigate the traffic and even block connections, it would simply produce a list of infected peers. Thus, mitigating P2P botnets clearly requires alternative approaches. However, as was discussed in Section 5.1, the majority of today's most prevalent malware families make use of client-server C&C. Thus, SQUEEZE can provide useful intelligence in the current threat landscape. Another issue is that botnets could use encrypted C&C protocols that are harder to detect at the network level. However, the specific mechanism used to model and detect C&C communication is largely orthogonal to our work. SQUEEZE could be combined with C&C detection techniques that do not suffer from this limitation, such as those based on behavioral detection at the network level [23], or on host-based analysis of information flows [26]. Strategy A, in particular, can be trivially combined with arbitrary C&C detection mechanisms.

A further limitation is that our tool can currently provide limited information on DGAs it triggers. That is, SQUEEZE can observe a number of requests going to generated domains, however it can neither detect that these are C&C servers (since they are typically offline) nor reveal how the DGA works. This limitation can be overcome by applying slicing and gadget extraction techniques [34] that can extract the DGA as a functional, self-contained piece of code that can be used to generate new domains. Finally, like most techniques based on the dynamic analysis of malware, SQUEEZE may be thwarted by malware that detects it is running in an instrumented environment and refuses to run. In recent years, a number of techniques have been proposed that attempt to mitigate this problem [35, 36, 37, 38, 39, 40].

7 Related Work

Botnets have been the subject of a significant amount of research. A survey of this topic has been performed by Bailey et al.[41]. Several studies have been performed with the goal of measuring and understanding the botnet phenomenon [4, 5, 6, 8, 9, 11, 12]. In other cases, researchers went a step further and actively infiltrated a botnet with the goal of obtaining further insight, taking down or even taking over the botnet [7, 3, 13, 14]. Another well-explored topic is the network-level detection of botnets, based on the bots' crowd-like behavior [42], on their reaction to C&C commands at the network [23] or host level [26], or on signatures generated by detecting recurrent patterns in botnet traffic [24, 25]. These approaches can be used either to create detection models that can be deployed to protect a network and detect which machines are already infected, or to detect and blacklist C&C servers. FIRE [20] identifies malware C&C servers and tracks their uptime, to identify networks that persistently host malicious activities. This has already led some ISPs to take an interest in the issue and improve their security practices [43]. SQUEEZE can be readily adapted to take advantage of any of these techniques for detecting C&C communication.

Limited coverage is a general problem of dynamic program analysis. Techniques to overcome this limitation have been proposed in

[5]http://maliciousnetworks.org

a number of fields such as software testings [44] and vulnerability discovery [45]. In the context of malware, Limbo [46] aims to recognize malicious device drivers (rootkits) by forcing execution to traverse the driver's control flow graph. Limbo however cannot ensure that the program state (that is, the values of registers and memory) is consistent with the program paths it is forcing. Other approaches [1, 47] overcome this limitation. Multi-path exploration [1] is the approach most closely related to our work, since it explores possible execution paths by modifying the execution environment before reverting to a snapshot. Multi-path exploration uses a constraint solver to find out how to modify the environment to lead execution down an alternative branch. MineSweeper [47] similarly aims to uncover trigger-based malware behavior using a combination of concrete and symbolic execution. While useful, these techniques suffer from the path explosion problem: The overall number of program paths that need to be explored grows exponentially. The reason is that, at each interesting branch in the program, the analysis has to follow two successor paths. Furthermore, code obfuscation can make the constraint solving step required by such tools provably hard [48]. SQUEEZE does not attempt to discover triggers for hidden behavior: Instead, it assumes that the behavior of interest (backup C&C communication) is triggered by the unavailability of network resources. This reduces the execution space that has to be explored to the (much smaller) endpoints tree, such as the one shown in Figure 2.

A different approach for increasing coverage is used by Reanimator [49]. By automatically identifying and modeling the code responsible for an observed behavior, Reanimator is able to recognize the same capability in other programs even if it is not observed at run-time. A limitation is that a behavior has to be triggered in at least one malware execution before it can be modeled.

8 Conclusion

Modern botnets are complex distributed systems designed to be resilient in the face of takedown attempts. To avoid losing control over their infected computers, botmasters use redundant C&C servers and failover C&C strategies such as domain generation algorithms. While it is possible to detect malware C&C servers by monitoring the execution of bots in a controlled environment, this approach suffers from limited coverage and will not reveal all C&C servers. Connections to backup C&C servers will not be triggered until the primary servers are taken down. In this paper we have introduced SQUEEZE, a system that uses a specialized form of multi-path exploration to trick bot binaries into revealing additional C&C endpoints and failover strategies. We introduced two alternative strategies for making use of domain knowledge on C&C communication to efficiently explore the execution paths that are revealed when communication with a contacted endpoint is allowed or blocked. By testing SQUEEZE on a diverse and representative malware dataset and comparing it against an ordinary analysis sandbox, we showed that it can reveal hundreds of additional active C&C servers and trigger DGA algorithms that bots use as a failover strategy.

9 Acknowledgements

The research leading to these results has received funding from the European Union Seventh Framework Programme under grant agreement n. 257007 (SysSec), from the Prevention, Preparedness and Consequence Management of Terrorism and other Security-related Risks Programme European Commission - Directorate-General Home Affairs (project i-Code), and from the Austrian Research Promotion Agency (FFG) under grant 820854 (TRUDIE). This publication reflects the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

10 References

[1] Moser, A., Kruegel, C., Kirda, E.: Exploring Multiple Execution Paths for Malware Analysis. In: IEEE Symposium on Security and Privacy. (2007)

[2] Decker, A., Sancho, D., Kharouni, L., Goncharov, M., McArdle, R.: A study of the Pushdo / Cutwail Botnet. `http://us.trendmicro.com/imperia/md/content/us/pdf/threats/securitylibrary/Study_of_pushdo.pdf` (2009)

[3] Stone-Gross, B., Cova, M., Cavallaro, L., Gilbert, B., Szydlowski, M., Kemmerer, R., Kruegel, C., Vigna, G.: Your botnet is my botnet: analysis of a botnet takeover. In: Proceedings of the 16th ACM conference on Computer and communications security (CCS). (2009)

[4] Chiang, K., Lloyd, L.: A case study of the rustock rootkit and spam bot. In: Proceedings of the First Workshop on Hot Topics in Understanding Botnets. (2007)

[5] Binsalleeh, H., Ormerod, T., Boukhtouta, A., Sinha, P., Youssef, A., Debbabi, M., Wang, L.: On the analysis of the zeus botnet crimeware toolkit. In: International Conference on Privacy, Security and Trust. (2010)

[6] Grizzard, J.B., Sharma, V., Nunnery, C., Kang, B.B.H., Dagon, D.: Peer-to-Peer Botnets: Overview and Case Study. In: Proceedings of the First Workshop on Hot Topics in Understanding Botnets table of contents. (2007)

[7] Holz, T., Steiner, M., Dahl, F., Biersack, E., Freiling, F.: Measurements and mitigation of peer-to-peer-based botnets: a case study on storm worm. In: Proceedings of the 1st Usenix Workshop on Large-Scale Exploits and Emergent Threats. (2008)

[8] Dittrich, D., Dietrich, S.: P2p as botnet command and control: a deeper insight. In: 3rd International Conference On Malicious and Unwanted Software (Malware). (2008)

[9] Stock, B., Goebel, J., Engelberth, M., Freiling, F.C., Holz, T.: Walowdac - analysis of a peer-to-peer botnet. In: Proceedings of the 2009 European Conference on Computer Network Defense (EC2ND). (2009)

[10] Fitzgibbon, N., Wood, M.: Conficker. C: A technical analysis. `http://www.sophos.com/sophos/docs/eng/marketing_material/conficker-analysis.pdf` (2009)

[11] Thomas, K. an Nicol, D.: The koobface botnet and the rise of social malware. In: 5th International Conference On Malicious and Unwanted Software (Malware). (2010)

[12] Kang, B.B., Chan-Tin, E., Lee, C.P., Tyra, J., Kang, H.J., Nunnery, C., Wadler, Z., Sinclair, G., Hopper, N., Dagon, D., Kim, Y.: Towards complete node enumeration in a peer-to-peer botnet. In: Proceedings of the 4th International Symposium on Information, Computer, and Communications Security (ASIACCS). (2009)

[13] Cho, C.Y., Caballero, J., Grier, C., Paxson, V., Song, D.: Insights from the inside: a view of botnet management from infiltration. In: USENIX Workshop on Large-Scale Exploits and Emergent Threats (LEET). (2010)

[14] Stone-Gross, B., Holz, T., Stringhini, G., Vigna, G.: The Underground Economy of Spam: A Botmaster's Perspective of Coordinating Large-Scale Spam Campaigns. In: USENIX Workshop on Large-Scale Exploits and Emergent Threats (LEET). (2011)

[15] Mushtaq, A.: Smashing the Mega-d/Ozdok botnet in 24 hours. `http://blog.fireeye.com/research/2009/11/smashing-the-ozdok.html` (2009)

[16] Krebs, B.: Takedowns: The Shuns and Stuns That Take the Fight to the Enemy. McAfee Security Jouranl (6) (2010)

[17] Cranton, T.: Cracking Down on Botnets. `http://blogs.technet.com/b/microsoft\_blog/archive/2010/02/25/cracking-down-on-botnets.aspx` (2011)

[18] Krebs, B.: Researchers Kneecap Pushdo Spam Botnet. `http://krebsonsecurity.com/2010/08/researchers-kneecap-pushdo-spam-botnet/` (2010)

[19] Boscovich, R.: Taking Down Botnets: Microsoft and the Rustock Botnet. `http://blogs.technet.com/b/microsoft\_blog/archive/2011/03/17/taking-down-botnets-microsoft-and-the-rustock-botnet.aspx` (2011)

[20] Stone-Gross, B., Moser, A., Kruegel, C., Almaroth, K., Kirda, E.: FIRE: FInding Rogue nEtworks. In: Annual Computer Security Applications Conference (ACSAC). (2009)

[21] Bayer, U., Kruegel, C., Kirda, E.: TTAnalyze: A Tool for Analyzing Malware. In: Proceedings of the 15th European Institute for Computer Antivirus Research (EICAR 2006) Annual Conference. (2006)

[22] Willems, C., Holz, T., Freiling, F.: Toward Automated Dynamic Malware Analysis Using CWSandbox. IEEE Symposium on Security and Privacy **5**(2) (2007)

[23] Wurzinger, P., Bilge, L., Holz, T., Goebel, J., Kruegel, C., Kirda, E.: Automatically generating models for botnet detection. In: Proceedings of the 14th European conference on Research in computer security (ESORICS). (2009)

[24] Perdisci, R., Lee, W., Feamster, N.: Behavioral Clustering of HTTP-Based Malware and Signature Generation Using Malicious Network Traces. In: USENIX conference on Networked Systems Design and Implementation (NSDI). (2010)

[25] Rieck, K., Schwenk, G., Limmer, T., Holz, T., Laskov, P.: Botzilla: detecting the "phoning home" of malicious software. In: Proceedings of the 2010 ACM Symposium on Applied Computing (SAC). (2010)

[26] Jacob, G., Hund, R., Holz, T., Kruegel, C.: JACKSTRAWS: Picking Command and Control Connections from Bot Traffic. In: USENIX Security Symposium. (2011)

[27] Bayer, U., Milani Comparetti, P., Kruegel, C., Kirda, E.: Scalable, Behavior-Based Malware Clustering. In: Network and Distributed System Security (NDSS). (2009)

[28] Bellard, F.: QEMU, a Fast and Portable Dynamic Translator. In: USENIX Annual Technical Conference. (2005)

[29] Roesch, M.: Snort: lightweight intrusion detection for networks. In: USENIX Systems Administration Conference (LISA). (1999)

[30] Mushtaq, A.: World's Top Malware. `http://blog.fireeye.com/research/2010/07/worlds_top_modern_malware.html` (2010)

[31] Williams, J.: Bredolab Takedown, Another Win for Collaboration. `http://blogs.technet.com/b/mmpc/archive/2010/10/26/bredolab-takedown-another-win-for-collaboration.aspx` (2010)

[32] Rashid, F.Y.: Harnig Botnet Goes Offline After Rustock Raid. `http://www.eweekeurope.co.uk/news/harnig-botnet-goes-offline-after-rustock-raid-25588` (2011)

[33] Wilcoxon, F.: Individual Comparisons by Ranking Methods. Biometrics Bulletin **1**(6) (1945) 80–83

[34] Kolbitsch, C., Holz, T., Kruegel, C., Kirda, E.: Inspector Gadget: Automated Extraction of Proprietary Gadgets from Malware Binaries. In: IEEE Symposium on Security and Privacy. (2010)

[35] Chen, X., Andersen, J., Mao, Z.M., Bailey, M., Nazario, J.: Towards an Understanding of Anti-Virtualization and Anti-Debugging Behavior in Modern Malware. In: IEEE International Conference on Dependable Systems and Networks (DSN). (2008)

[36] Dinaburg, A., Royal, P., Sharif, M., Lee, W.: Ether: Malware Analysis via Hardware Virtualization Extensions. In: ACM Conference on Computer and Communications Security (CCS). (2008)

[37] Kang, M.G., Yin, H., Hanna, S., McCamant, S., Song, D.: Emulating Emulation-Resistant Malware. In: Proceedings of the 2nd Workshop on Virtual Machine Security (VMSec). (2009)

[38] Balzarotti, D., Cova, M., Karlberger, C., Kruegel, C., Kirda, E., Vigna, G.: Efficient Detection of Split Personalities in Malware. In: Network and Distributed System Security Symposium (NDSS). (2010)

[39] Johnson, N.M., Caballero, J., Chen, K.Z., McCamant, S., Poosankam, P., Reynaud, D., Song, D.: Differential Slicing: Identifying Causal Execution Differences for Security Applications. In: IEEE Symposium on Security and Privacy. (2011)

[40] Lindorfer, M., Kolbitsch, C., Milani Comparetti, P.: Detecting Environment-Sensitive Malware. In: Recent Advances in Intrusion Detection (RAID). (2011)

[41] Bailey, M., Cooke, E., Jahanian, F., Xu, Y., Karir, M.: A survey of botnet technology and defenses. Conference For Homeland Security, Cybersecurity Applications and Technology (2009) 299–304

[42] Gu, G., Zhang, J., Lee, W.: Botsniffer: Detecting botnet command and control channels in network traffic. In: Proceedings of The 15th Annual Network and Distributed System Security Symposium (NDSS 2008). (2008)

[43] Krebs, B.: Naming and Shaming 'Bad' ISPs. `http://krebsonsecurity.com/2010/03/naming-and-shaming-bad-isps/` (2010)

[44] Godefroid, P., Klarlund, N., Sen, K.: Dart: directed automated random testing. In: Proceedings of the 2005 ACM SIGPLAN conference on Programming language design and implementation. PLDI 05 (2005) 213–223

[45] Cadar, C., Ganesh, V., Pawlowski, P.M., Dill, D.L., Engler, D.R.: Exe: automatically generating inputs of death. In: Proceedings of the 13th ACM conference on Computer and communications security. CCS 06 (2006) 322–335

[46] Wilhelm, J., Chiueh, T.: A Forced Sampled Execution Approach to Kernel Rootkit Identification. In: Symp. on Recent Advances in Intrusion Detection (RAID). (2007)

[47] Brumley, D., Hartwig, C., Liang, Z., Newsome, J., Poosankam, P., Song, D., Yin, H.: Automatically identifying trigger-based behavior in malware. In: Botnet Detection. (2008)

[48] Sharif, M.I., Lanzi, A., Giffin, J.T., Lee, W.: Impeding malware analysis using conditional code obfuscation. In: Network and Distributed System Security (NDSS). (2008)

[49] Milani Comparetti, P., Salvaneschi, G., Kirda, E., Kolbitsch, C., Kruegel, C., Zanero, S.: Identifying Dormant Functionality in Malware Programs. In: IEEE Symposium on Security and Privacy. (2010)

Distilling Critical Attack Graph Surface iteratively through Minimum-Cost SAT Solving

Heqing Huang, Su Zhang, Xinming Ou
Kansas State University
Manhattan, KS, USA
{cleanini, zhangs84, xou}@ksu.edu

Atul Prakash and Karem Sakallah
University of Michigan
Ann Arbor, MI, USA
{aprakash, karem}@umich.edu

ABSTRACT

It has long been recognized that it can be tedious and even infeasible for system administrators to figure out critical security problems residing in full attack graphs, even for small-sized enterprise networks. Therefore a trade-off between analysis accuracy and efficiency needs to be made to achieve a reasonable balance between completeness of the attack graph and its usefulness. In this paper, we provide an approach to *attack graph distillation*, so that the user can control the amount of information presented by sifting out the most critical portion of the full attack graph. The user can choose to see only the k most critical attack paths, based on specified severity metrics, *e.g.* the likelihood for an attacker to carry out certain exploit on certain machine and the chance of success. We transform an dependency attack graph into a Boolean formula and assign cost metrics to attack variables in the formula, based on the severity metrics. We then apply Minimum-Cost SAT Solving (MCSS) to find the most critical path in terms of the least cost incurred for the attacker to deploy multi-step attacks leading to certain crucial assets in the network. An iterative process inspired by Counter Example Guided Abstraction and Refinement (CEGAR) is designed to efficiently guide the MCSS to render solutions that contain a controlled number of realistic attack paths, forming a *critical attack graph surface*. Our method can distill critical attack graph surfaces from the full attack graphs generated for moderate-sized enterprise networks in only several minutes. Experiments on various sized network scenarios show that even for a small-sized critical attack graph surface (around 15% the size of the original full attack graph), the calculated risk metrics are good approximation of the values computed with the full attack graph, meaning the distilled critical attack graph surface is able to capture the crucial security problems in an enterprise network for further in-depth analysis.

Categories and Subject Descriptors

H.4 [**Information Systems Applications**]: Miscellaneous; D.2.8 [**Software Engineering**]: Metrics—*complexity measures, performance measures*; C.2.0 [**General**]: [Security and protection]; K.6.5 [**Security and protection**]: [Unauthorized access][Management of computing and information system]; D.2.8 [**Software science**]: [Metrics]

1. INTRODUCTION

Attack graph generation has become a mature technology and generating attack graphs for enterprise networks of realistic sizes is now practical [1, 14, 15, 29, 32]. However, the complexity of the generated attack graphs is usually beyond human analysis without proper visualization [12, 27]. Moreover, our previous work shows that downstream analysis based on attack graphs, such as quantitative risk assessment, incur substantial performance penalty for large attack graphs if a precise result is desired. Though precise calculation and complete information does have its advantages, efficiency does become a bottleneck. Intuition indicates that for analyses such as quantitative risk assessment, it is not necessary to use the full attack graphs since it is possible that a small portion of the critical vulnerabilities constitutes the most risk for the full graph.Furthermore, through identifying the most critical part correctly, the same approximation range of the risk metrics can be obtained by analyzing only the prioritized portion of the full attack graph.Therefore we believe in order to bring the analysis of generated attack graphs into a more practical level and also help the system administrator (SA) perform in-depth analysis, reaching a balance point for the trade-off between the completeness and efficiency of attack graph analysis is a necessary step.

1.1 Our solution

Our method is to iteratively distill *critical attack graph surfaces* from a full attack graph, by leveraging an off-the-shelf Minimum-Cost SAT solver (MCSS). A critical attack graph surface is a subgraph of the original attack graph that represents most critical security problems in the system. MCSS can help sift out the minimum-cost solution from a full attack graph, which from the attacker's view is the minimum-effort attacking route to conquering certain critical assets in the network. Moreover, we perform it in an iterative and flexible way, so that critical attack paths are distilled from the full attack graph, generating critical attack graph surfaces on demand for further in-depth analysis.

The iterative approach controls the completeness of the information by outputting a portion of the attack graph that contains just the first k critical attack paths for a user-provided k. Our basic idea is inspired by the Counter-Example Guided Abstraction and Refinement (CEGAR) [3] method on hardware and software analysis. A dependency attack graph has a similar structure as a logic gate circuit composed of only AND and OR gates. Boolean Satisfiability (SAT) solvers are common components to be leveraged in CEGAR verification, *e.g.*, through the Satisfiability Modulo Theories (SMT) solvers [6]. Even SAT is NP-hard, the current branch and bound SAT solvers are capable of handling Boolean formulas arising from lots of practical problems with millions of clauses and literals. Therefore transforming a dependency attack graph into a Boolean formula and using carefully designed refinement method and cost assignment strategy to guide MCSS iteratively is expected to be an efficient method. An alternative option is to implement the k-shortest hyperpath algorithm [26], which does have a polynomial-time algorithm. We choose the SAT solving-based approach, in that the transformed Boolean formula can be further extended to support more flexible analysis. For example, our approach provides the flexibility to define the SAT problem according to the SA's analysis priorities; thus only the critical attack paths related to the analysis needs will be rendered. This provides a convenient way to define and configure the logic structure to conduct various types of customized analysis. For the efficiency aspect of producing the k-shortest path attack surface, we will leave the comparison of the two approaches as future work.

1.2 Our contributions

Backward induction on attack graphs to create Boolean formulas.

Formalization and transformation of a dependency attack graph into a Boolean formula in conjunctive normal form (CNF) or disjunctive normal form (DNF) has been proposed before [10, 28, 37]. In our work we perform the transformation in a back manner, which initiates from the goal states and grounds at the attacker's initial location. In this way, all the attacking steps' logic dependencies are captured within a CNF formula by using this novel interpretation of attack graph. We provide detailed explanation in Section 3.2.

Leveraging MCSS to prioritize attack paths.

To the best of our knowledge, we are the first to explore an iterative approach based on MCSS to repetitively prioritize the first k critical attack paths and then generate critical attack graph surface from the collected paths. With the distilled critical attack graph surface, we bring the analysis and visualization of attack graph to a more practical level.

Flexible on-demand critical attack graph surface generation.

Various on-demand critical attack graph surfaces can be generated via user specified input Boolean formulae. Moreover, we also explore various ways to extract counter examples from both unrealistic and realistic attack segments as "lemmas" used in iterative refinement. By repetitively augmenting the lemma database, problems are updated for the MCSS to more efficiently generate critical attack graph surface.

Evaluation of the methodology on efficiency, scalability and accuracy.

We performed a range of controlled experiments to check the efficiency and scalability of our method. The results show that by producing critical attack-graph surfaces of around 15% of the size of the original graph efficiently, one can achieve very good approximation of quantitative risk assessment result, and thus dramatically improve the scalability of attack graph in-depth analysis.

2. RELATED WORK

Attack graph is a technique proposed in late 1990's [5, 31] and fully developed in the first decade of 2000's [15, 18, 29, 35]. Various approaches have been proposed to use attack graphs to conduct further analysis about network security. Some of the approaches apply graph algorithms to identify critical portions of attack graphs or find optimal hardening options [16, 28]. There have also been applications of stochastic reasoning and other AI methods on attack graphs [7, 23, 33]. More recently, various methods for deriving quantitative security metrics from attack graphs have been proposed [8, 36]. Our work provides another tool in the arsenal of attack graph-based analysis. Our contribution is the concept of attack graph surface and a way to compute the critical attack graph surface iteratively through SAT solving. The approach is flexible and can produce critical attack graph surface according to a number of user-defined criteria, which can then be used in combination with the other attack graph-based analysis methods, such as quantitative security risk assessment, to make them more efficient and focused on the core problems in the network.

SAT solving has been applied in network configuration management [25]. MinCostSAT has been applied to attack graph for context-aware security management [10]. Boolean formula-based approach has also been used to identify minimum-cost network hardening options from attack graphs [37]. Our application of SAT solving on attack graphs is different, in that it focuses on identifying critical problems in an attack graph, instead of directly identifying hardening solutions. Therefore the way Boolean formulas are created from attack graphs is different from these prior works. We believe differentiating severity among attack steps and identification of critical attack surface is an important problem and can benefit a number of other analysis, including the identification of the most effective hardening options.

The notion of attack surface [13, 20, 21] was proposed in software engineering as a metric to indicate the exposed resources to potential abuses. Our notion of attack graph surface is inspired by the basic ideas behind attack surface, and refers to a portion of an attack graph that demonstrates the most critical security exposure in a networked system.

3. METHODOLOGY

3.1 Methodology framework

The framework of our method of iterative distillation of attack graphs is shown in Figure 1. The components included in the dotted rectangle except for the MinCostSAT Solver (MCSS), are all designed and implemented in this work. Our distillation method goes through several core steps iteratively, implemented in various modules. First, we transform the full attack graph into a CNF Boolean formula ϕ, and then based on related SA's requirements logic constraints are conjoined with ϕ, after which we use the CVSS [24] *Access Complexity* metrics to derive attacker's cost metric for literals in the resulting formula.[1] After this preparation, the constrained and cost-related formula is fed into a MinCostSAT solver as the initial SAT problem. Each round the solution from the solver is refined and verified, from which counter-example constraints may be generated or a realistic critical attack path is chained logically, and then in both case, the refined solution will be generalized into "lemmas" used to subsequently augment ϕ. The SAT problem is updated incrementally with the learned lemmas for the next round. Beside that, the verified realistic attack path will be added to the attack graph surface, which is gradually sifted out from the original full attack graph.

[1]Our use of CVSS to derive the cost metrics is due to the fact that CVSS metrics are maintained and publicly available in the National Vulnerability Database (NVD). However, we realize that CVSS also has its limitation to be used in assigning attacker cost. We believe a more systematic method is needed to derive and calibrate the cost metrics and it is an orthogonal problem for future research.

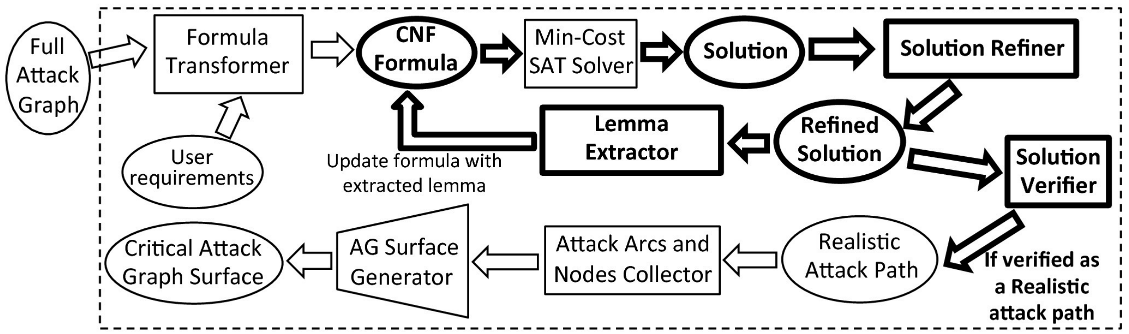


Figure 1: Iterative critical attack graph surface distillation framework

Formula Transformer.

With the input, a full attack graph, our Boolean formula transformer converts important attack steps into Boolean clauses through a depth-first traversal (DFS) of the full attack graph. The complexity of this conversion is linear to the number of nodes and arcs in the full attack graph, as only constant operations are performed during DFS along each node and arc. Detailed description of the transformation can be found in Section 3.2. The converted formula contains the logical structure of the attack graph, forming a conjunctive normal form (CNF). Initial cost metrics are derived from CVSS for attack steps involving exploiting a vulnerability; specific cost metrics related to particular exploits could also be configured by user manually as for different scenarios the user may want to adjust the cost metric based on network-specific historical data. Moreover, the user could also provide on-demand requirements by adding constraints into the original CNF formula to reconfigure the logical structure of the full attack graph so as to perform more fine-grained and user-specified analysis. For example, requirements for certain critical assets could be specified as attacker's goals by the SA, so that a critical attack graph surface related to the specified goals are extracted from the full attack graph according to SA's security concerns. Or exploits of certain vulnerabilities can be excluded since they will be patched soon. This is similar to the SA asking questions like: "What is the critical attack surface related to assets A and B?", or "What is the critical attack surface without considering vulnerability X?" The above could also be done during the initial attack graph generation. However, combining this with critical attack graph surface generation provides a convenient way to highlight the various security problems on the fly.

MinCostSAT Solver.

With the transformed CNF formula, which contains both the logical constraints of the full attack graph and the user on-demand requirements, the MinCostSAT Solver (MCSS) can produce a minimum-cost solution in each round. However, not all solutions correspond to valid attack paths. The solution may include some irrelevant zero-cost nodes or even contains a cycle and makes the whole solution unrealistic. The solution refiner and verifier deal with these imperfections in solutions.

Solution Refiner and Verifier.

The solver may assign some zero-cost literals to be *True* that are irrelevant to the critical attack path connected through other *True* assignments in the solution. The solution refiner eliminates these irrelevant literals from the solution through a traversal on the full attack graph guided by the solution. The complexity is linear to the number of *True* literals in the solution. Only the *True* literals included in the traversal path are included as the critical attack path. Moreover, the solution may also contain unrealistic path segments. The solution verifier checks whether the solution is indeed a "realistic" attack path, which must initiate from one of the attacker's initial locations to one of the goal node.

Lemma Extractor.

The idea of counterexample generation is to extract the root causes for a solution to be invalid as a constraint and negate the constraint to form a "lemma" so that the solver will not return such solutions in the future.

Critical Attack Arcs and Nodes Collector and Attack Graph Surface Generator.

The refinement loop highlighted in the figure is processed iteratively until the first k most critical paths have been stored, or the SAT solver returns an UNSAT solution indicating no more paths available. Then a critical attack graph surface could be generated based on the collected arcs and nodes from only the top "k" attack paths.

3.2 SAT problem formalization

3.2.1 Full attack graph backward induction

The logical relationships represented in a dependency attack graph could be analyzed in two directions. One is from the attacker's initial location to the final goal state, which we call forward deduction. This has been used [10] to encode the effect of changing configurations on attackability. The second is backward induction, which is from the attack goal state to the initial attacker location. This has also been used in transforming an attack graph into disjunctive normal form (DNF) [37] that captures the exact configuration conditions for a privilege(s) to be obtainable. In our work, we perform a novel backward induction, which produces a conjunctive normal form (CNF) that encodes the requirement relations between attack steps. After binding cost metrics to literals in the CNF formula, a cost sensitive SAT solver could be leveraged to rank the attack paths to reach a goal.

In Figure 2, a toy example of dependency attack graph is rendered to help review the logical structure of the graph and provide examples for the transformation process. An OR node (the diamond-shaped node) represents the privilege an attacker can gain after exploiting one of its predecessor AND node, *e.g.* p_1 can be achieved through one of e_1, e_2, e_4 and e_5, corresponding to the OR logic. An AND node (the round-shaped node) is used to simulate the process of exploiting a vulnerability on a host, after all its predecessors are obtained. A special type of AND node is called a *grounding node*, which has no OR node as its predecessor and supported by all SOURCE nodes, *e.g.* AND node e_2. SOURCE nodes

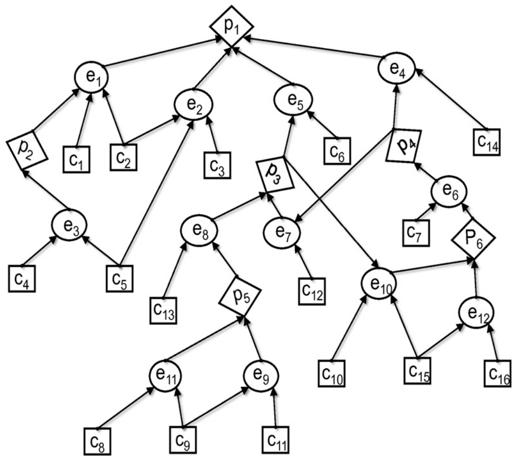


Figure 2: A toy example for dependency Attack Graph

represent the configuration or vulnerability fact about the current network, which is known to be *True* in the network. e_2 is logically supported by all the grounding facts that indicates e_2 could be triggered initially without any prerequisite privilege. However, for e_1, one privilege p_2 should be required in addition to grounding facts c_1 and c_2. This means the attacker should have already gained the privilege p_2 in order to carry out the exploit e_1. Since all the SOURCE nodes in the attack graph are used to represent existing facts about the network, they are considered "True" and thus are not actually encoded in our CNF formula. For example, two attack paths for reaching the final goal privilege p_1 through exploits e_1, e_2 and e_3 could be succinctly traced and expressed below:

$$e_3 \rightarrow p_2 \rightarrow e_1 \rightarrow p_1;\ e_2 \rightarrow p_1$$

For the OR node, (*e.g.* p_1 in the figure), our rationale behind this backward transformation is that, for the goal privilege p_1 to be obtained, which attack step can be taken by the attacker to conquer p_1? We perform this backward induction iteratively on all the AND and OR nodes in the graph to link all the possible attack steps which could logically contribute to exploiting a given privilege, and store the backward causality relations into a CNF formula for further automatic analysis. Four predecessors of p_1: e_1, e_2, e_4 and e_5, are all the options that could be considered during backward induction. In order to acquire privilege p_1, at least one of the exploits from the set of p_1's predecessor nodes should be carried out successfully by the attacker. We define the symbol *Cons()* to indicate the basic logic constraint encoded based on the original attack graph. For p_1, we have

$$Cons(p_1) \quad ::= \quad p_1 \Rightarrow e_1 \vee e_2 \vee e_4 \vee e_5$$

Or, equivalently,

$$Cons(p_1) \quad ::= \quad \neg p_1 \vee e_1 \vee e_2 \vee e_4 \vee e_5$$

And for a normal AND node like e_1

$$Cons(e_1) \quad ::= \quad e_1 \Rightarrow p_2 \wedge c_1 \wedge c_2$$

Or, equivalently,

$$Cons(e_1) \quad ::= \quad (\neg e_1 \vee p_2) \wedge (\neg e_1 \vee c_1) \wedge (\neg e_1 \vee c_2)$$

As SOURCE nodes c_1 and c_2 are always enabled, and being set *True*, the transformed constraints for e_1 could be reduced as

$$Cons(e_1) \quad ::= \quad (\neg e_1 \vee p_2) \wedge \textit{True} \wedge \textit{True}$$

For this type of exploits, the logic constraint indicates that in order to set e_1 to be "True", its predecessor privilege p_2 should be set "True" as well.

Based on this transformation, each OR node and AND node will be represented by one or more disjunctive clauses. By conjoining all these constraints we obtain a CNF formula, ϕ, that encodes the whole backward logical dependency of the attack graph. For this toy example we could build ϕ like this,

$$\begin{aligned}\phi ::= & Cons(p_1) \wedge Cons(e_1) \wedge Cons(p_2) \wedge Cons(e_5)\\ & \wedge Cons(p_3) \wedge Cons(e_8) \wedge Cons(p_5) \wedge Cons(e_7)\\ & \wedge Cons(p_4) \wedge Cons(e_6) \wedge Cons(p_6) \wedge Cons(e_4)\end{aligned}$$

Note that the $groundingnodes$ do not show up in the formula, since they are all reduced to *True*. This transformation process is linear in the size of the attack graph, because the process needs to traverse each arc in the graph exactly once to build the formula.

Through solving the SAT problem enforced in ϕ, which describes backward logical dependency of the full attack graph, local attack steps are connected into multi-step attack paths, which initiate from attacker starting location and end at one of the attack goal states.

3.2.2 Formula-based attack scenario update

Usually, the concern from the system administrator (SA) is related to the critical assets (modeled as goal nodes in attack graphs) that should be protected in the network, or SA may be concerned that some hardening or risk management strategy could impact the whole network, *e.g.* patching certain vulnerabilities or blocking certain network connections may disrupt services. Hence, the SA wants to update the attack scenario described in attack graph for various analysis interests and security concerns. We design goal node-related constraints and reconfiguration constraints to help SA update these specific concerns based on the initial Boolean formula.

For multi-goal scenario, we use a constraint to make the critical assets (the goal states) stand out. We simply symbolize the goal-node constraints as $goalCons(goalNodes)$. For the example attack graph, if we want to configure both p_1 and p_2 to be the goal node, we only have to insert a goal constraint like

$$goalCons(p_1, p_2) \quad ::= \quad (p_1 \vee p_2)$$

to the ϕ. That means, at least one of the goal nodes p_1 or p_2 should be $True$ in the final solution.With this flexible initialization for $goalCons$, the SA could define a specific problem which only contains a set of goal nodes that he/she is concerned about, and later by iteratively calling the SAT solver, a critical attack graph surface related to that on-demand set of goal nodes could be rendered.

For the network reconfiguration specification, the SA could disable certain attack steps by setting negations for variables related to vulnerabilities or network reachability, in order to simulate vulnerabilities patching or network reconfiguration. We symbolize the reconfiguration constraints as $reconfCons(andNodes)$. For the toy attack graph example, if the SA has defined the network reachability modeled in e_9 to be blocked and also the vulnerabilities related to the exploit e_5 and e_8 to be patched, a reconfiguration constraint

$$reconfCons(e_9, e_8, e_5) \quad ::= \quad (\neg e_9 \wedge \neg e_8 \wedge \neg e_5)$$

should be inserted to the ϕ. In this case, both the attack path

$$e_9 \rightarrow p_5 \rightarrow e_8 \rightarrow p_3 \rightarrow e_5 \rightarrow\ p_1$$

$$e_{11} \rightarrow p_5 \rightarrow e_8 \rightarrow p_3 \rightarrow e_5 \rightarrow\ p_1$$

is negated from the solution space. These can all be done through an interactive user interface for the attack graph.

Table 1: CVSS Access Complexity Metric maps to Probability

AC metric	*Description*	*Prob*
High	Specialized access conditions exist.	0.2
Medium	The access conditions are somewhat specialized.	0.6
Low	Specialized access conditions or extenuating circumstances do not exist.	0.9

3.3 Leverage MinCostSAT solver

3.3.1 Minimum-Cost SAT solving

Since we want to prioritize attack paths based on their severity, MinCostSAT is used so that a solution (related to an attack path) always has the minimum cost. The cost function for Boolean variables can be set up so that lower cost corresponds to higher severity. The optimization is from an attacker's perspective: we want to find a path that minimizes the cost for an attacker to obtain certain privileges in a network, based on the CNF formula that enforces privilege/exploit dependencies. MinCostSAT has been thoroughly studied by the SAT solving community [4, 9, 17, 22]. Even if it is an NP-hard problem, modern cost-sensitive SAT solvers have been considerably successful in dealing with Boolean formulas arising from practical problems. It has been reported that formulas with millions of variables and clauses can be solved in seconds [9].

A cost-sensitive solver, the MinCostChaff solver is chosen in our method, [9] which is developed based on the zChaff SAT solver [19] by Princeton University. Given a Boolean formula ϕ with n literals $a_1, a_2, a_3 \ldots, a_n$, cost vector, $\{c_i | c_i \geq 0, 1 \leq i \leq n\}$, the solver will provide a set of assignments to the set of literals $X_i \in \{0,1\}^n$ that minimizes the function:

$$C = \sum_{i=1}^{n} c_i X_i$$

The literals assigned value 1 is considered *True*. As long as meaningful numerical cost values can be assigned to each attack-step Boolean variable, the MinCostChaff solver provides a solution with the minimum cost of function C, indicating the attack path with the minimum cost for the attacker. Moreover, always returning the minimum-cost solution helps eliminate lots of redundant attack steps a cost-insensitive solver tends to include, since extra attack steps will increase the cost.

3.3.2 Cost vector aggregation

An important concern in any quantitative analysis is where to obtain the numeric parameter input. In our prior work, we map the CVSS Access Complexity Metric to a probability indicating the likelihood of success in exploiting the vulnerability, which is then used in performing quantitative risk assessment [11]. The Common Vulnerability Scoring System (CVSS) [24, 34] provides an open framework for communicating the characteristics and impacts of IT vulnerabilities. CVSS consists of three metric groups: Base, Temporal and Environmental. Each of this group produces a vector of compressed textual representation that reflects various properties of the vulnerability (*metric vector*). A formula takes the vector and produces a numeric score in the range of 0 to 10 indicating the severity of the vulnerability. An important contribution of CVSS is the metric vector: it provides a wide range of vulnerability properties that are the basis for deriving the numerical scores.

We have used the Access Complexity (AC) from the CVSS metric vector to derive probability of attack success when calculating quantitative risk metrics [30]. Table 1 illustrates the possible values of the AC metric and the mapped probabilities (*Prob* column). In this work, we turn the probability into a cost used in the MinCost-SAT solver. The formula for converting the probability to the cost value is as follows:

$$exploitingCost = 10 * (1.00 - sqrt(\mathit{Prob}))$$

When the success likelihood is 1, the cost is 0. When the likelihood is 0, the cost is maximum (10). We take an square root of the probability to indicate the fact that when the success likelihood increases, the cost decreases unevenly — quicker at the beginning and slower at the end. For each specific enterprise network, the SA could also assign system-related cost vectors, based on historical data. How to assign such cost values is an important research question that is beyond the scope of this paper.

Even assuming some variations might occur, while differentiating ways of assigning the cost vector, our experiments in Section 4.2 show that the whole distilled critical attack graph surface will not vary too much, and also the risk metrics computed from the critical attack graph surface will converge to nearly the same status after reaching around 15% the size of the full attack graph.

3.4 Attack graph surface generation through lemma extraction

The core idea behind our iterative attack graph surface generation is inspired by CounterExample-Guided Abstraction and Refinement (CEGAR) [3]. In CEGAR, a solution returned from a solver indicates a potential flaw in the system being analyzed. It either represents a real flaw, or is a *spurious solution* because the analysis model *abstracts* the concrete systems by simplifying assumptions that may not always hold in the concrete system. Thus the solution needs to be checked against the concrete system to determine whether it is a realistic solution, or a spurious one. If it is a spurious solution, the solver needs to be informed why this is the case so it will not output such solutions in the future. From the counterexample, *i.e., the spurious solution*, one can extract a *lemma* that explains why the solution is not valid in the concrete system, in the form of a logical formula. The lemma is then fed back to the solver to become part of the constraints in the original formula. The solver is called again and the whole process repeats, gradually *refining* the formula, until the solution returned from the solver constitutes a real flaw, or no solution can be found in which case no flaw exists in the system.

The idea of CEGAR is useful in attack graph analysis because the input to the attack graph is necessarily an imprecise view of the system. Some relevant information like the fine-grained conditions under which an exploit is possible may not always be provided in the input due to the feasibility of obtaining all such information up front. For this reason the attack graph, and the boolean formula derived from it, can be considered an abstraction of the system and not all paths in it (solutions to the formula) represent realistic attacks. The desirable feature of CEGAR is that one does not need to obtain all the fine-grained details about attack conditions up front, but only needs to include information *relevant* to the system under analysis through refuting counter examples and generalizing lemmas.

3.4.1 Imperfect solutions from a SAT solver

The imperfectness of the solutions returned by the SAT solver arises from two main reasons: 1) existence of zero-cost nodes; 2) existence of unrealistic path segments. Thus the solutions require further refinement and verification. In the first case the solver may arbitrarily assign a few zero-cost literals to be *True*, which will not contribute to any real path. We can refine the solution by eliminating these irrelevant nodes through a solution-guided depth-first traversal (SGDFT) of the full attack graph. The number of traversal steps is linear to the number of literals being set *True* by the solver. An additional benefit of SGDFT is to order the *True* literals into an attack path, since the solver does not return the solutions in the path order. Even after this solution refinement process, the solution is not guaranteed to be a realistic attack path, due to the possibility of cycles. We define the criteria to verify a solution as a realistic attack path: the solution initiates from an attacker initial location and ends at one of the goal nodes, representing one of the predefined critical assets. According to the criteria, an ordered depth-first traversal (ODFT) is performed to verify the solution against the criteria. Based on SGDFT and ODFT, either a realistic critical attack path could be recorded, or an invalid path will be obtained for further bogus path segments negation.

3.4.2 Iterative lemma extraction

To render the critical attack graph surface, the problem defined by ϕ should be dynamically augmented to repetitively guide the MCSS to answer the question "What is the next critical attack path?" Therefore, no matter whether the solution is verified as realistic or not, we will extract a more general portion from the refined solution. The conjunction of the Boolean literals of the corresponding portion is negated to create a counter-example constraint, which is conjoined with the formula ϕ. In this way, the updated formula will not output "similar" attack paths as the current solution. This both enables refuting invalid attack paths, and ensuring the top *k* critical attack paths can be generated iteratively. How to extract the solution precisely to maximize the elimination of the undesired portion in the solution space becomes critical for the efficiency of our approach. Therefore, extraction rules should be able to help guide the solver perform iterative real critiacl attack path generation accurately and efficiently. This is the important criteria for extraction rules definition.

We design a special type of constraints, namely counter-example constraints, denoted as $counterCons()$, in order to guide the MinCostSAT solver to iteratively and efficiently search for the critical solutions. The parameter of $counterCons()$, is "the minimum unsatisfiable path segments" inspired by the Minimum Unsatisfiable Subset (MUS) concept used in the CEGAR method [2]. MUS is considered an efficient method to speed up the refinement process in CEGAR method by eliminating useless iterations on similar spurious violations.

Minimum Unsatisfiable Path Segments (MUPS).

The path segments considered as the minimum unsatisfiable portion could be extracted from the returned solution, based on a combination of user-selected extraction rules, in order to provide a stronger refinement to save more iterations for valid solution and to build various flexible critical attack graph surface.

Based on different strategies to extract and generalize the path sequences to form MUPS and negate the extracted path segment in order to learn a "lemma" to add into ϕ iteratively, various refinement methods are available. In order to improve the efficiency and accuracy of our approach, several strategies on how to generalize the major "lemma" have been tested. Then, we define two basic extraction rules based on the mentioned criteria, and several optional ones are also being discussed for flexible utility of our approach:

Basic Extraction Rule 1.

Whenever a solution is checked as an unrealistic attack path, (*e.g.* the loopy solution or redundant solution), we perform a depth-first search on the full attack graph for the path segment and extract nothing more than the minimum unsatisfiable path segment (MUPS) in the solution to produce a lemma. Extracting more generalized lemmas can help further cut down the solution space, but we should be aware that critical and user-concerned security problem should not be eliminated. One example is loopy segments in a solution. When there are cycles in the attack graph, a satisfying solution to our transformed formula may end up chaining some attack steps into a cycle. For instance, a solution which contains the path $p_3 \rightarrow e_{10} \rightarrow p_6 \rightarrow e_6 \rightarrow p_4 \rightarrow e_7 \rightarrow p_3 \rightarrow e_5 \rightarrow p_1$ in Figure 2, will be a valid solution to the Boolean formula, but a spurious one as an attack path. In this case, we will detect and extract the loopy path segment:

$$p_3 \rightarrow e_{10} \rightarrow p_6 \rightarrow e_6 \rightarrow p_4 \rightarrow e_7 \rightarrow p_3$$

as a more generalized pattern from the solution and negate the cycle to produce a counterexample constraint:

$$\begin{aligned} &\textit{counterCons(MUPS)} \\ &\quad ::= \neg(e_{10} \wedge p_6 \wedge e_6 \wedge p_4 \wedge e_7 \wedge p_3) \\ &\quad ::= \neg e_{10} \vee \neg p_6 \vee \neg e_6 \vee \neg p_4 \vee \neg e_7 \vee \neg p_3 \end{aligned}$$

This lemma will prevent other similar spurious solutions form being output. For instance,

$$p_4 \rightarrow e_7 \rightarrow p_3 \rightarrow e_{10} \rightarrow p_6 \rightarrow e_6 \rightarrow p_4 \rightarrow e_4 \rightarrow p_1$$

will never be output by the solver once the above lemma has been added. Therefore a properly defined MUPS could ensure the counterexample is general enough, so that no more similar "bogus" paths will be considered as solutions. The strong refinement based on the generalized "lemma" can eliminate more on the search space and avoid many unnecessary iterations.

Basic Extraction Rule 2.

Whenever a solution is checked as a realistic attack path, we consider the whole attack path as a MUPS. By negating the whole attack path to produce the "lemma", the MinCostSAT solver will search for the next minimum-cost solution based on the updated formula. In order to match the SA's interests on outputting identical critical attack path segments more efficiently, by excluding similar or the same critical path segments, we define "Exclude-one" extraction rule as an optional rule for rule 2, which excludes the very last node, namely the attacker's initial location state from the realistic attack path and negate the rest of the path. This means any longer paths which contain the identical critical path segment, but launched from other initial states or through other non-initial locations will be eliminated. For instance, when the shorter realistic attack path: $e_{12} \rightarrow p_6 \rightarrow e_6 \rightarrow p_4 \rightarrow e_4 \rightarrow p_1$ has been returned, we will extract the whole path but exclude the grounding node e_{12}, which means other longer paths based on the critical path segment from p_6 to p_1, *e.g.* path: $e_{11} \rightarrow p_5 \rightarrow e_8 \rightarrow p_3 \rightarrow e_{10} \rightarrow p_6 \rightarrow e_6 \rightarrow p_4 \rightarrow e_4 \rightarrow p_1$, will not appear. The assumption here for this optional rule is as long as there is already a lower cost realistic attack path leading to the goal node being output, other "tedious" longer paths with an extra "tail" path segment attached to the critical shorter path segment are not considered worthy of further analysis.

Other optional extraction rules.

More interactive extraction rules could be selected by the user, according to their specific concern of the security problem. A user may think certain combinations of complicate and low-possibility exploits are unrealistic in a given scenario when considering the specific level of adversary skills. For instance, while analyzing zero-day vulnerabilities' impact on the network, the user may believe that almost no adversaries for this system are capable of leveraging three zero-day vulnerabilities in different applications to deploy an attack. Then a lemma could be defined to say: for any realistic attack path that contains a combination of three zero-day vulnerabilities in the three applications, the path will not be considered. Such a lemma can be easily expressed as a boolean constraint to use in our system. Other extraction rules based on various criteria could also be implemented in similar manners, providing flexibility in analysis.

3.5 Algorithm of Iterative attack graph surface distillation

Algorithm 1 Pseudocode for iterative critical attack graph distillation

```
Input: $G_f${Full attack graph};
       $k${Number of critical attack paths desired}
Output: $G_s${Critical attack graph surface};
        $\psi${a collection of ordered attack paths}

Initiate user specified distillation requirements, including extra
extraction rules $optionRules$.

$\phi \leftarrow$formulaTransformer($G_f$)
$\phi \leftarrow (\phi \wedge goalCons() \wedge reconfCons(); costVector)$
pathNum $\leftarrow k$
while $pathNum > 0$ do
    if MinCostSATSolver($\phi$) returns SAT solution $\chi$ then
        $\chi \leftarrow$solutionRefiner($\chi$)
        isRealpath$\leftarrow$solutionVerifier($\chi$)
        if isRealpath then
            pathNum $\leftarrow$ pathNum - 1
            store $\chi$ store in $\psi$ as critical attack path
            store $\delta \leftarrow$ criticalArcsNodesCollector($\chi$)
            {$\delta$: arcs & nodes to generate critical AG surface.}
            $p \leftarrow$ MUPS_Extrator($\chi, Rule2, optionRules$)
            {$p$: Minimal Unsatifiable Path Segments}
        else
            $p \leftarrow$ MUPS_Extrator($\chi, Rule1, optionRules$)
        end if
        $counterCons(p) \leftarrow$ counterexampleGenerator($p$)
        {$counterCons(p)$: restored lemma.}
        $\phi \leftarrow \phi \wedge counterCons(p)$
    else
        Break
        {UNSAT returned from MinCostSAT solver, no more
        attack path for the updated SAT problem}
    end if
end while
$G_s \leftarrow$ AG_surface_Generator($\delta$)
```

Algorithm 1 describes the whole iterative process for our distillation method, which reflects the basic requirements from the user. Due to space constraint, we only describe the high-level procedure for our approach on iterative attack graph surface distillation. Note that the algorithm always terminates since in each round one path segment is found and negated to update ϕ. Since there are only a finite number of attack paths in the input graph, either we will find k valid attack paths, or we will exhaust all path segments in the system (UNSAT returned from the SAT solver). In either case the algorithm will terminate. At the end of the algorithm, the critical attack graph surface is generated by outputting the sub-graph that only contains the collected critical attack paths' arcs and vertices.

4. EXPERIMENTS AND RESULTS

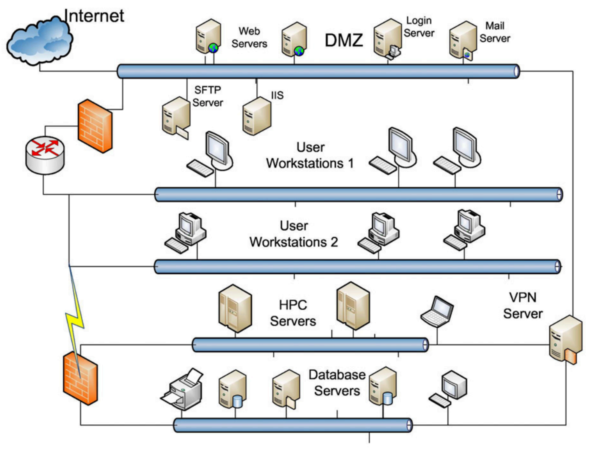


Figure 3: An example enterprise network

We conducted several groups of experiments to test important aspects of our method, including efficiency, scalability, and accuracy. Most of the experiments were performed on an enterprise network scenario adapted from a real network, except for a range of simulated network scenarios used in scalability test. Figure 3 shows the adapted enterprise network used in our experiments. The network includes five subnets: a DMZ (Demilitarized Zone) that hosts a number of publicly accessible servers, two internal subnets used for user workstations, a high-performance computing servers subnet, and a database servers subnet. The total number of hosts is around five hundred and for hosts with identical configuration, a grouping method is performed to create a summarized input file for the attack graph generator. The general firewall configuration allows the Internet to visit the servers in DMZ through the respective ports; some of the hosts in the two workStation subnets can reach the database servers and the HPC servers. Critical assets are defined by the system administrator (SA). The SA must ensure that the HPC servers and Web Servers are functioning well and also the data on the Database servers are well protected. Besides these critical components in the enterprise network, we also consider the VPN server as one of the attack goal, since this server acts as a tunnel for setting up private connections from Internet. If compromised, malicious connections could be launched from Internet to other critical assets in the network. Basically in this adapted network, we define the attacker goals as the VPN server, the HPC servers, the Web servers and the Database servers.

4.1 Efficiency and scalability analysis

Table 2 records the numbers of literals and clauses in the transformed CNF formula for each of the full attack graphs generated from four network scenarios. Data of test case C and D are collected according to our adapted network shown in Figure 3; we set goal nodes as Database servers and HPC servers for C and D respectively to test the efficiency aspect. Data of test case A and B are generated based on two simulated large networks with thousands of hosts distributed in tens of subnets to analyze the scalabil-

Table 2: Scale of Test Cases

Test Case	A	B	C	D
# literal	40680	21086	292	280
# clauses	41158	21274	333	321

ity of our method and also the effects of our basic extraction rules (Section 3.4).

We consider the number of literals and clauses used to initialize the CNF formula a good metric to reflect the complexity of the corresponding network scenarios' attackability, since literals (representing exploits or privileges) and clauses (containing the logic relation between exploits and privileges) encode the main logic structure of a full attack graph. Note that the clauses here only include the constraints like $Cons()$, and not other iteratively augmented constraints such as $counterCons()$. The full attack graph is generated after some grouping on host is already done. As a result hosts with similar configurations will not produce redundant attack nodes, which is the reason why the formula only contains hundreds of literals and clauses.

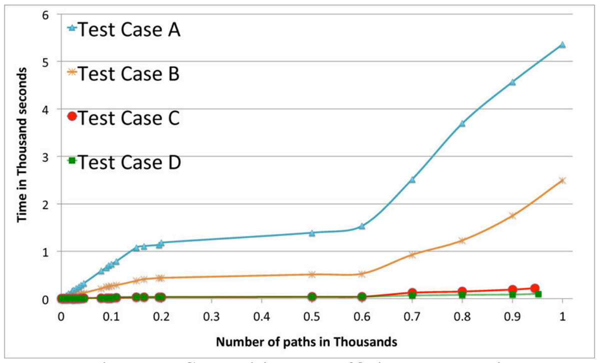


Figure 4: Scalability and efficiency analysis

The X-axis in Figure 4 represents the number of paths (in thousands) for each distilled attack graph surface of the corresponding scenario. The Y-axis shows the time (in thousand seconds) for producing an attack graph surface that contains the corresponding number of critical attack paths. The test cases C and D demonstrate how efficient our method is in dealing with moderate-sized network. Case C took 221 seconds to render 945 critical attack paths and case D took 102 seconds to render the first 952 critical attack paths.

From the test cases on scalability (A and B) we can see that the initial portion of the curves grows much faster. This means more time is required to search a real attack path at the initial stage. According to our observation of the real attack path distributions in the MinCostChaff solution space from all these experiments, the density of "fake" path segments contained in solutions is much higher in the first two hundred rounds. Thus it takes longer time to search for a real attack path initially. Then the time used for distilling each real attack paths comes to a more stable and efficient stage. This indicates that the initial steep curve is used mostly for learning the lemmas needed to identify what is a real attack path. Once enough lemmas are learned, the attack-graph surface generation can be performed much more quickly.

Starting at around six-hundredth path, the curves become steep again, which is because a new set of longer "fake" loopy path segments appeared in the solutions with longer paths. Fortunately, paths beyond this stage do not contribute significantly to the network's risk, as shown in Section 4.2. This real attack path density variation trend shows the effect of the extraction of "lemmas" from those "fake" paths: the space of the fake paths are trimmed down largely at the beginning and the critical real attack paths can be rendered efficiently.

Simply using the number of paths to gauge the size of the created attack graph surface may not be a very precise one. Therefore we collected data (in Table 3) to analyze the relationship among size related metrics of all the created attack graph surfaces. We pick two Size Levels with the distilled attack graph surfaces 20% and 50% of the full attack graphs in terms of number of attack paths. In general, the average percentage of each set of metric indicates they are comparatively close, but the percentage of arcs and vertices (represent the actually size of the attack graph) in both Size Level cases are smaller than the percentage in terms of the critical attack paths.

Table 3: Size Metrics

Size Related Metrics	Size Level 1	Size Level 2
% on number of Attack Paths	20%	50%
% on number of Arcs	17.97%	43.54%
% on number of Vertices	16.76%	39.17%

Generally speaking, for the scalability test cases, A and B with around 40000 and 20000 clauses, our method takes no more than 5500 and 3000 seconds respectively, to distill critical attack graph surface with around 1000 real attack paths for each huge scenario. Our experiments on accuracy (Section 4.2) showed that for risk analysis of the attack graph, an attack graph surface with a portion around 15% of the largest attack graph surface is big enough to capture the major and crucial security problems in the given network.

4.2 Accuracy analysis based on risk assessment of attack graph

4.2.1 Risk assessment analysis

In our past work we have developed a quantitative risk assessment method based on attack graphs [11]. Each attack-step node is associated with a conditional probability based on the nature of the exploit. The conditional probability indicates the likelihood the attack step can be successfully carried out when the attacker has obtained the necessary pre-conditions (network access, *etc*.). In most cases these probabilities are derived from the CVSS metrics [24]. In the cost assignment discussed in Section 3.3.2, the probability is also used to derive an attacker cost for our critical attack surface generation. We now show that a small critical attack graph surface contains the most risk in the network, through applying the quantitative risk assessment algorithm on the attack graph surfaces of various sizes.

4.2.2 Accuracy analysis based on risk assessment

For the accuracy validation, we carried out four groups of comparison experiments to analyze the risk assessment result trends based on the gradually increased distilled attack graph surface, using four different attack goals in the same network scenarios depicted in Figure 3: DB servers, HPC servers, VPN server and WebServers. Also, for each of the group, two sets of critical attack surfaces based on *both* CVSS cost assignment and uniform cost assignment (labeled "even" in the diagram) are sifted out and risk assessment is performed on each set of critical attack surfaces. The purpose of using the "even" cost assignment is to test the sensitivity of our method on different numerical binding to literals used in MinCostSAT solver.

Figure 5 shows the data we collected for each set of the experiments. The X-axis represents the number of attack paths in the distilled attack graph surface (in thousands) and the Y-axis is the risk assessment value, ranging from 0 to 1. For the four group and eight trend curves, we observe that in general the risk values increase with the growth of the attack-surface size, which means the more

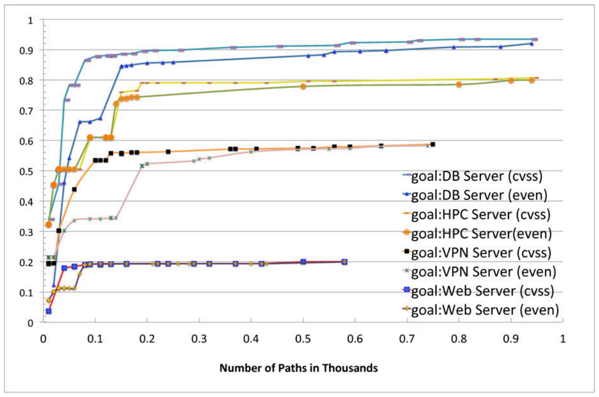


Figure 5: Risk assessment result trend

attack paths contained in an attack graph surface, the more risk is discovered corresponding to the scenario. Moreover, the trend shows that when the attack graph surfaces contain around 15% of the attack paths in the biggest attack graph surfaces, the risk value already converge to a good approximation value of that of the full graph. That means, our method is able to capture the major security problem residing in the network, through distilling a critical surface which is a small but crucial portion of the full attack graph.

For each set of the experiments we also observe that the lines of CVSS-based cost assignment reach the approximation point much faster than the uniform cost-based ones, and lines based on both cost assignments converge to nearly the same level of risk value after the number of attack graph surface reaches around 25% of the biggest attack surface. This indicates that variations on risk assessment results caused by different cost assignment strategy for the comprehensive critical attack graph surface can be mitigated by slightly increasing the size of the attack surface.

Generally speaking, the result shows that while the distilled attack graph surface reaches a comparatively small portion of the full attack graph, the risk assessment value will approach a good approximation of the full attack graph, which means the loss of accuracy is controlled within a tolerable range, provided a reasonable number of critical paths are sifted out (less than 15% of the size of full attack graph). Observation of the collected critical attack paths shows that the critical surface tends to retain those attack routes with comparatively few direct steps of attacks, or longer paths based on some easily triggered vulnerabilities. The tedious attack routes constituting of very hard exploits have been tagged with low priority. Only if the system administrator (SA) really wants to analyze some longer and harder attack routines, could our method render those less critical attack paths. In this way, the SA could gradually analyze different prioritized portions of the attack graph in terms of the critically levels.

5. DISCUSSION

With the critical attack graph surface method, a number of security analysis methods become available. For example, the system administrator (SA) might desire to pay more attention to several specific vulnerabilities on certain hosts, or certain types of firewall policies configured in the network. In that case, by conjoining corresponding literals of these vulnerabilities and configurations with formula ϕ, critical attack paths only related to these specified attack nodes could be sifted out. We could even set hypothetical zero-day vulnerabilities on crucial hosts and configure these vulnerabilities in ϕ, then we analyze the impact of them by distilling critical attack paths that contain one or more introduced zero-days. Identifying more useful constraints to fit various analysis methods is an interesting topic, which requires a deep understanding of how the SA behave in network security analysis and what the most desired security problems he/she wants to figure out. Building a more configurable platform for SA's interactive analysis, could guide the SAT solver return more relevant portion of the attack graph. We will leave this as future work.

6. CONCLUSION

We have proposed an iterative attack graph distillation method based on minimum-cost SAT solving, to create a critical attack graph surface that highlights the most important security problems in a computer network. Our method is inspired by Counter Example-Guided Abstraction and Refinement. The transformation of an attack graph into a boolean formula provides a flexible and extensible method to render different views of the attack graph's critical surface, and thus improves its utility. Through flexibly setting the number of critical attack paths to be sifted out for further in-depth analysis, the trade-off between efficiency and accuracy can be balanced. Our experiments on risk assessment validate the assumption that a small portion of the attack graph captures the most important problems in the system, and our method can efficiently identify this portion as the critical attack graph surface, which is usually less than 15% of the full attack graph's size in our experiments. This efficient and accurate method of attack graph distillation can be used in combination with further in-depth attack graph analysis, such as quantitative risk assessment, which would have been computational expensive on a full attack graph. The system also provide a extensible framework for conducting logical analysis on attack graphs using defined logic formula encoding rules and more extensible optional lemma extraction rules.

7. ACKNOWLEDGMENTS

We would like to thank Zhaohui Fu and Sharad Malik for providing the code and assistance for running the MinCostChaff solver. This material is based upon work supported by U.S. National Science Foundation under grant no. 1038366 and 1018703, AFOSR under Award No. FA9550-09-1-0138, and HP Labs Innovation Research Program. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation, AFOSR, or Hewlett-Packard Development Company, L.P.

8. REFERENCES

[1] P. Ammann, D. Wijesekera, and S. Kaushik. Scalable, graph-based network vulnerability analysis. In *Proceedings of 9th ACM Conference on Computer and Communications Security*, Washington, DC, November 2002.

[2] Z. S. Andraus, M. H. Liffiton, and K. A. Sakallah. Refinement strategies for verification methods based on datapath abstraction. In *Proceedings of the 2006 Asia and South Pacific Design Automation Conference(ASP-DAC)*, 2006.

[3] E. Clarke, O. Grumberg, S. Jha, Y. Lu, and H. Veith. Counterexample-guided abstraction refinement. In *Proceedingsof Computer Aided Verification (CAV)*, volume 1855, pages 154–169, 2000.

[4] O. Coudert. On solving covering problems. In *33rd Design Automation Conference (DAC'96)*, pages 197–202, 1996.

[5] M. Dacier, Y. Deswarte, and M. Kaâniche. Models and tools for quantitative assessment of operational security. In *IFIP SEC*, 1996.

[6] L. de Moura and N. Bjørner. Z3: An efficient SMT solver. In *Tools and Algorithms for the Construction and Analysis of System (TACAS), pages 337-340*, 2008.

[7] R. Dewri, N. Poolsappasit, I. Ray, and D. Whitley. Optimal security hardening using multi-objective optimization on attack tree models of networks. In *14th ACM Conference on Computer and Communications Security (CCS)*, 2007.

[8] M. Frigault and L. Wang. Measuring network security using Bayesian network-based attack graphs. In *Proceedings of the 3rd IEEE International Workshop on Security, Trust, and Privacy for Software Applications (STPSA'08)*, 2008.

[9] Z. Fu and S. Malik. Solving the minimum-cost satisfiability problem using sat based branch and bound search. In *Proceedings of the International Conference on Computer-Aided Design (ICCAD'06)*, San Jose, CA, USA, 2006.

[10] J. Homer and X. Ou. SAT-solving approaches to context-aware enterprise network security management. *IEEE JSAC Special Issue on Network Infrastructure Configuration*, 27(3), April 2009.

[11] J. Homer, X. Ou, and D. Schmidt. A sound and practical approach to quantifying security risk in enterprise networks. Technical report, Kansas State University, 2009.

[12] J. Homer, A. Varikuti, X. Ou, and M. A. McQueen. Improving attack graph visualization through data reduction and attack grouping. In *The 5th International Workshop on Visualization for Cyber Security (VizSEC)*, 2008.

[13] M. Howard, J. Pincus, and J. Wing. Measuring relative attack surfaces. In *Proceedings of Workshop on Advanced Developments in Software and Systems Security*, 2003.

[14] K. Ingols, R. Lippmann, and K. Piwowarski. Practical attack graph generation for network defense. In *22nd Annual Computer Security Applications Conference (ACSAC)*, Miami Beach, Florida, December 2006.

[15] S. Jajodia, S. Noel, and B. O'Berry. Topological analysis of network attack vulnerability. In V. Kumar, J. Srivastava, and A. Lazarevic, editors, *Managing Cyber Threats: Issues, Approaches and Challanges*, chapter 5. Kluwer Academic Publisher, 2003.

[16] S. Jha, O. Sheyner, and J. M. Wing. Two formal analyses of attack graphs. In *Proceedings of the 15th IEEE Computer Security Foundations Workshop*, pages 49–63, Nova Scotia, Canada, June 2002.

[17] X. Y. Li. *Optimization Algorithms for the Minimum-Cost Satisfiability Problem*. PhD thesis, North Carolina State University, Raleigh, North Carolina, 2004.

[18] R. P. Lippmann, K. W. Ingols, C. Scott, K. Piwowarski, K. Kratkiewicz, M. Artz, and R. Cunningham. Evaluating and strengthening enterprise network security using attack graphs. Technical Report ESC-TR-2005-064, MIT Lincoln Laboratory, October 2005.

[19] Y. S. Mahajan, Z. Fu, and S. Malik. Zchaff2004: An efficient SAT solver. In *Lecture Notes in Computer Science SAT 2004 Special Volume*, pages 360–375. LNCS 3542, 2004.

[20] P. Manadhata and J. Wing. An attack surface metric. *IEEE Transactions on Software Engineering*, 2010.

[21] P. Manadhata, J. Wing, M. Flynn, and M. McQueen. Measuring the attack surfaces of two FTP daemons. In *Proceedings of the 2nd ACM workshop on Quality of Protection*, 2006.

[22] V. M. Manquinho and J. ao P. Marques-Silva. Search pruning techniques in SAT-based branch-and-bound algorithmsfor the binate covering problem. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 21:505–516, 2002.

[23] V. Mehta, C. Bartzis, H. Zhu, E. Clarke, and J. Wing. Ranking attack graphs. In *Proceedings of Recent Advances in Intrusion Detection (RAID)*, September 2006.

[24] P. Mell, K. Scarfone, and S. Romanosky. *A Complete Guide to the Common Vulnerability Scoring System Version 2.0*. Forum of Incident Response and Security Teams (FIRST), June 2007.

[25] S. Narain, G. Levin, S. Malik, and V. Kaul. Declarative infrastructure configuration synthesis and debugging. *Journal of Network and Systems Management*, 2008.

[26] L. R. Nielsena, K. A. Andersenb, and D. Pretolani. Finding the k shortest hyperpaths. *Computers & Operations Research*, 32:1477–1497, 2005.

[27] S. Noel and S. Jajodia. Managing attack graph complexity through visual hierarchical aggregation. In *VizSEC/DMSEC '04: Proceedings of the 2004 ACM workshop on Visualization and data mining for computer security*, pages 109–118, New York, NY, USA, 2004. ACM Press.

[28] S. Noel, S. Jajodia, B. O'Berry, and M. Jacobs. Efficient minimum-cost network hardening via exploit dependency graphs. In *19th Annual Computer Security Applications Conference (ACSAC)*, December 2003.

[29] X. Ou, W. F. Boyer, and M. A. McQueen. A scalable approach to attack graph generation. In *13th ACM Conference on Computer and Communications Security (CCS)*, pages 336–345, 2006.

[30] X. Ou and A. Singhal. *Quantitative security risk assessment of enterprise networks*. Information Security. SpringerBrief, 2011.

[31] C. Phillips and L. P. Swiler. A graph-based system for network-vulnerability analysis. In *NSPW '98: Proceedings of the 1998 workshop on New security paradigms*, pages 71–79. ACM Press, 1998.

[32] D. Saha. Extending logical attack graphs for efficient vulnerability analysis. In *Proceedings of the 15th ACM conference on Computer and Communications Security (CCS)*, 2008.

[33] R. Sawilla and X. Ou. Identifying critical attack assets in dependency attack graphs. In *13th European Symposium on Research in Computer Security (ESORICS)*, Malaga, Spain, October 2008.

[34] M. Schiffman, G. Eschelbeck, D. Ahmad, A. Wright, and S. Romanosky. *CVSS: A Common Vulnerability Scoring System*. National Infrastructure Advisory Council (NIAC), 2004.

[35] O. Sheyner, J. Haines, S. Jha, R. Lippmann, and J. M. Wing. Automated generation and analysis of attack graphs. In *Proceedings of the 2002 IEEE Symposium on Security and Privacy*, pages 254–265, 2002.

[36] L. Wang, T. Islam, T. Long, A. Singhal, and S. Jajodia. An attack graph-based probabilistic security metric. In *Proceedings of The 22nd Annual IFIP WG 11.3 Working Conference on Data and Applications Security (DBSEC'08)*, 2008.

[37] L. Wang, S. Noel, and S. Jajodia. Minimum-cost network hardening using attack graphs. *Computer Communications*, 29:3812–3824, November 2006.

RIPE: Runtime Intrusion Prevention Evaluator

John Wilander
Dept of Computer Science,
Linköpings universitet
john.wilander@gmail.com

Nick Nikiforakis
Katholieke Universiteit Leuven
Belgium
nick.nikiforakis@cs.kuleuven.be

Yves Younan
Katholieke Universiteit Leuven
Belgium
yves.younan@cs.kuleuven.be

Mariam Kamkar
Dept of Computer Science,
Linköpings universitet
mariam.kamkar@liu.se

Wouter Joosen
Katholieke Universiteit Leuven
Belgium
wouter.joosen@cs.kuleuven.be

ABSTRACT

Despite the plethora of research done in code injection countermeasures, buffer overflows still plague modern software. In 2003, Wilander and Kamkar published a comparative evaluation on runtime buffer overflow prevention technologies using a testbed of 20 attack forms and demonstrated that the best prevention tool missed 50% of the attack forms. Since then, many new prevention tools have been presented using that testbed to show that they performed better, not missing any of the attack forms. At the same time though, there have been major developments in the ways of buffer overflow exploitation.

In this paper we present *RIPE*, an extension of Wilander's and Kamkar's testbed which covers 850 attack forms. The main purpose of RIPE is to provide a standard way of testing the coverage of a defense mechanism against buffer overflows. In order to test RIPE we use it to empirically evaluate some of the newer prevention techniques. Our results show that the most popular, publicly available countermeasures cannot prevent all of RIPE's buffer overflow attack forms. ProPolice misses 60%, LibsafePlus+TIED misses 23%, CRED misses 21%, and Ubuntu 9.10 with non-executable memory and stack protection misses 11%.

Categories and Subject Descriptors

D.4.6 [**Operating Systems**]: Security and Protection—*Invasive software*; D.2.8 [**Software Engineering**]: Metrics—*Product metric*

1. INTRODUCTION

Buffer overflows are probably the single most well-known exploitation technique in the history of computer security. The ability to take control of the execution flow of a process by overwriting adjacent data, first received world-wide attention through the Morris worm [43] and since then has been used as the exploitation mechanism in most of the well-known worms (e.g. CodeRed [29] and SQLSlammer [28]) as well as in countless attacks against popular software. Due to the severity and popularity of this attack, researchers have produced a significant amount of papers describing techniques which can, among others, detect, defend, stop and heal buffer-overflows at all possible stages of a program's lifetime. A small number of these suggested techniques have reached production level, by being included in popular programming frameworks and operating systems.

Despite the amount of research conducted in the area of buffer overflows in specific, and code injection techniques in general, modern software is still plagued by buffer-overflows which are discovered, almost weekly, in one of the many popular software products. At the time of this writing, the US-CERT [47] reports that over 200 buffer-overflow vulnerabilities have been found in 2011, many of which are in products of Microsoft, Adobe and Google. This shows that the problem of buffer overflows is far from resolved and that researchers in both academia and industry should focus their efforts in creating countermeasures that can eventually be part of real running systems.

While there are standard ways to measure the performance overhead of a buffer overflow countermeasure, such as the SPEC CPU [44] and Olden benchmarks, there is no standard way of testing and comparing the defense coverage of any given countermeasure.

In 2003, Wilander and Kamkar [53] published a comparative evaluation on runtime buffer overflow prevention using a testbed of 20 attack forms and demonstrated that the best prevention tool missed 50% of the attack forms. That testbed has been used to demonstrate the effectiveness of subsequent tools and techniques [11, 14, 15, 35, 37, 46] and the outcome of the 2003 evaluation was used to motivate further preventive research [16, 27, 36, 40].

We believe that many of the attack techniques tested by that testbed are now outdated and thus a perfect "score" of a protection countermeasure against them is of limited value.

In this paper we introduce RIPE—*Runtime Intrusion Prevention Evaluator*—which comprises of 850 buffer overflow attack forms. The main purpose of RIPE is to quantify the protection coverage of any given countermeasure by performing a wide range of buffer-overflow attacks and recording their success or failure. The tool is released as free software (see Section 10 for availability) in an attempt to standardize the comparison between countermeasures and to further support research on code-injection countermeasures. In order to test the applicability and usability of the

RIPE testbed, we tested it against buffer overflow countermeasures that the authors have made publicly available or that were kindly provided to us.

This paper and the release of the RIPE testbed provides the following research contributions:

1. Implementation of the combinatorial set of buffer overflow attack forms built on 4 locations of buffers in memory, 16 target code pointers, 2 overflow techniques, 5 variants of attack code being executed, and 10 functions being abused. In total, 850 working attack forms.
2. Empirical evaluation of publicly available buffer overflow countermeasures using canaries, boundary checking, copying and checking target data, library wrapping, and non-executable memory.
3. Open source, fully documented testbed code and driver engine for evaluation and reporting.

The rest of this paper is structured as follows: The RIPE testbed is described in Sections 2 and 3. Section 4 gives an overview of buffer overflow prevention techniques. Our evaluation setup is explained in Section 5 and the results are presented in Section 6. Section 7 contains related work. Finally, Section 8 describes future work and we conclude in Section 9.

2. THE RIPE BUFFER OVERFLOW TESTBED

The RIPE (Runtime Intrusion Prevention Evaluator) testbed has a backend built in C and a frontend built in Python. The backend or "attack generator" of RIPE lets the user dynamically specify which type of buffer overflow she wants to test. For instance:

```
./ripe_attack_generator -f strcpy -t direct -l stack
-c ret -i nonop
```

... will perform the standard stack smashing attack abusing the `strcpy()` function to overflow a buffer on the stack all the way to the return pointer, redirecting it to injected attack code without a NOP sled. As another example:

```
./ripe_attack_generator -f sscanf -t indirect -l heap
-c funcptrstackparam -i returnintolibc
```

... will abuse the `sscanf()` function to perform an overflow of a buffer located on the heap, overwrite a general pointer and make it point to a function pointer parameter (indirect attack), and redirect that function pointer to return-into-libc attack code. In the next sections, we will present RIPE's *dimensions*, which are essentially all the user-configurable parameters of an attack. Then we'll enumerate all the parameters that build up the combined 850 attack forms.

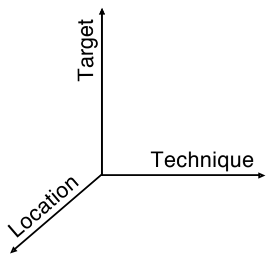
2.1 Testbed Dimensions

Wilander and Kamkar's testbed (hereafter referred to as the "NDSS'03 testbed") had three dimensions all of which are included in the new RIPE testbed too; *location* of buffer in memory, *target code pointer*, and *overflow technique* - Fig 1(a).

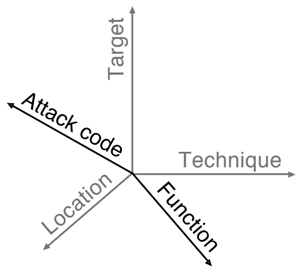
The new RIPE testbed has five dimensions including extended versions of the original three. The additions are *attack code* and *function abused* - Fig 1(b).

2.2 Dimension 1: Location

The first testbed dimension is the memory location of the buffer to be overflowed. Both the NDSS'03 testbed and RIPE support four buffer locations; Stack, Heap, BSS, and Data segment.



(a) Dimensions of NDSS'03 testbed



(b) Dimensions of RIPE testbed

Figure 1: The difference of dimensions supported by the NDSS'03 testbed and RIPE

2.3 Dimension 2: Target Code Pointer

The second testbed dimension is the target code pointer, i.e. the code pointer to redirect towards the attack code. RIPE supports the following target code pointers:

- *Return address*: The address stored by a function in order to return to the appropriate offset of the caller
- *Old base pointer*: The previous contents of the `EBP` register, which is used to reference function arguments and local variables
- *Function pointers*: Generic function pointers allowing programmers to dynamically call different functions from the same code
- *Longjmp buffers*: Setjmp/longjmp is a technique which allows programmers to easily jump back to a predefined point in their code (see Sec. 3.2).
- *Vulnerable Structs*: Structs which group a buffer and a function pointer and can be abused by attackers to overflow from one to the other (see Sec. 3.3).

With the exception of the *Return Address* and the *Old base pointer targets* that are Stack-specific, all other targets are allocated on all available data segments of a process, i.e. on the Stack (both as local variables and function arguments), on the Heap and on the Data/BSS segment.

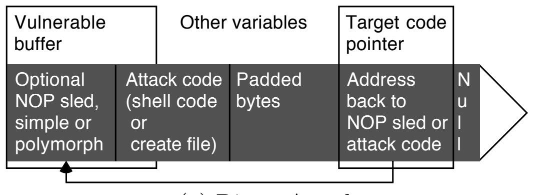
2.4 Dimension 3: Overflow Technique

The third testbed dimension is overflow technique. Both the NDSS'03 testbed and RIPE support *Direct* and *Indirect* overflowing techniques. In direct techniques, the target is adjacent to the overflowed buffer or can be reached by sequentially overflowing from the buffer. On the other hand, indirect overflowing makes use of generic pointers in a two-step approach. First the generic pointer is overflowed with the address of the target and then at a later dereference, the target is overwritten with attacker-controlled data. This technique was originally introduced by Bulba and Kil3r [10] as a way to bypass the StackGuard countermeasure. A pointer before the StackGuard canary was used to overwrite the return-address while maintaining the integrity of the canary.

2.5 Dimension 4: Attack Code

The fourth testbed dimension is new for RIPE – attack code. A user running the testbed can choose between attack code that spawns a shell on the vulnerable machine or attack code that creates a file in a specific directory. The former can be used when trying out individual attacks and the latter is used by the front-end part of RIPE which exhaustively tries all available attack combinations and then reports the full results. The variations of these two shellcodes are presented in the following list:

- *Shellcode without NOP sled*: This option can be useful in testing the accuracy of attacks as well as challenge countermeasures that rely on the detection of specific code patterns (such as the presence of a set of `0x90` bytes (NOP)) in the process' address space.
- *Shellcode with NOP sled*: This is the most-used form of shellcode that prepends the attacker's functionality with a set of NO-Operation instructions to improve the attacker's chances of correctly redirecting the execution-flow of the program into his injected code.
- *Shellcode with polymorphic NOP sled*: In this case, the NOP sled is not the standard set of `0x90` bytes but a set of instructions that can be executed without affecting the correctness of the actual attack code. As with the first variation, countermeasures that over-rely in the presence of standardized NOP-sleds will have difficulty countering such attacks. Akritidis et al. [1] conducted a study where, among others, they showed how obfuscation and encryption can be used by attackers to evade Network Intrusion Detection Systems (NIDS).
- *Return-into-libc*: Return-into-libc are attacks where the attacker does not inject new code in the process' address space but rather uses existing functions to perform his attack, for instance using the `system` libc-function to execute an interactive shell. This attack was essentially a natural evolution for attackers, when countermeasures that disallowed execution from writable memory pages, e.g. Data Execution Prevention (DEP) and $W \oplus$X, became popular in modern operating systems. RIPE uses `system()` for the spawning of an interactive shell and `creat()` for creating new files.
- *Return-Oriented Programming (ROP)*: Return-oriented programming [38] is the most recent way of carrying-out exploitation, once an attacker has achieved control of the execution flow. ROP is a generalization of Return-into-libc where now an attacker can use chunks of functionality from existing code (gadgets) and combine them to create new functionality. While we have implemented a ROP attack in our testbed, we haven't yet implemented stack-pivoting techniques and thus we can only trigger such an attack when we control the contents of the existing stack, as is the case in a stack-smashing attack.


Vulnerable buffer
Other variables
Target code pointer
Optional NOP sled, simple or polymorph
Attack code (shell code or create file)
Padded bytes
Address back to NOP sled or attack code
N u l l
(a) Direct Attack

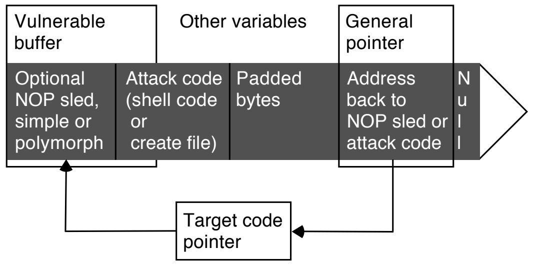

Vulnerable buffer
Other variables
General pointer
Optional NOP sled, simple or polymorph
Attack code (shell code or create file)
Padded bytes
Address back to NOP sled or attack code
N u l l
Target code pointer
(b) Indirect Attack

Figure 2: A direct and an indirect overflow payload with injected attack code

2.6 Dimension 5: Function Abused

The fifth and final testbed dimension is also new for RIPE – function abused. A user can choose to perform the buffer overflow with `memcpy()`, `strcpy()`, `strncpy()`, `sprintf()`, `snprintf()`, `strcat()`, `strncat()`, `sscanf()`, `fscanf()`, and also with "homebrew", a loop-based equivalent of `memcpy`.

The `n`-containing functions are designed to take the target buffer size into account which should prevent buffer overflows. The size however, is provided by the developer (static or calculated) and thus a miscalculation can cancel-out the protection offered by these functions. Known caveats include the fact that parameter `n` means *total* buffer size for `strncpy()` but *remaining* buffer space for `strncat()` [25], and if `n` is undefined for instance because of a `NULL` value in the length calculation `strncpy()` will allow for buffer overflow as shown in CVE-2009-4035 [22]:

```
line1 = getNext(line);  // May return NULL
if ((n = line1 - line) > 255) {
  n = 255;
}
strncpy(buf, line, n);  // n undef or < 0
```

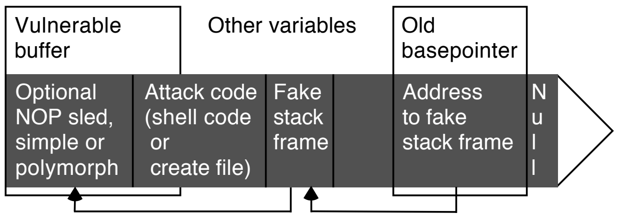
3. BUILDING PAYLOADS

RIPE's attack generator dynamically builds the specified payload and performs the attack on itself, i.e., the code contains all the required vulnerable buffers and pointers as well as the logic for offsets, attack code and overflows.

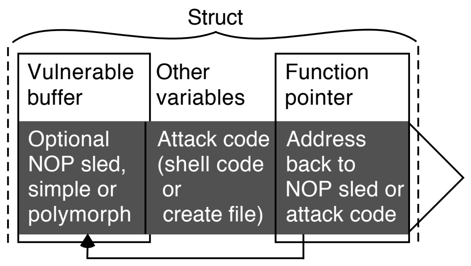
Figure 2(a) shows the payload of a direct overflow with injected code, and Figure 2(b) shows an *indirect* overflow using an intermediate pointer to target the code pointer.

3.1 Fake Stack Frame

The old base pointer is pushed on the stack immediately above the return address and is a possible target code pointer. The overflow redirects the base pointer towards an injected fake stack frame with a fake return pointer pointing to the attack code - Fig. 3(a).


Vulnerable buffer
Other variables
Old basepointer
Optional NOP sled, simple or polymorph
Attack code (shell code or create file)
Fake stack frame
Address to fake stack frame
N u l l

(a) Fake Stack Frame


Struct
Vulnerable buffer
Other variables
Function pointer
Optional NOP sled, simple or polymorph
Attack code (shell code or create file)
Address back to NOP sled or attack code

(b) Overflowed struct

Figure 3: An attack where a fake stack frame (with an attacker-controlled return address) is created and an overflow of a function pointer from within the same struct

3.2 Longjmp Buffer

Longjmp in C allows the programmer to explicitly jump back to functions, not going through the chain of return addresses. Consider a program where function A first calls `setjmp()`, then calls function B which in turn calls function C. If C now calls `longjmp()` the control is directly transferred back to function A, popping both C's and B's stack frames of the stack. Longjmp buffers contain the environment data required to resume execution from the point `setjmp()` was called. This environment data includes a base pointer and a program counter. If the program counter is redirected to attack code the attack will be executed when `longjmp()` is called.

3.3 Struct With Function Pointer

A struct containing a buffer and a function pointer can allow for an internal buffer overflow attack within the struct since there is no reordering of variables to make the code pointer unreachable from the buffer, nor are there any canary values between the buffer and the target code pointer - Fig 3(b). Such structs have been previously discussed by Zhivich et al.[54].

4. RUNTIME BUFFER OVERFLOW PREVENTION

The research in countering buffer overflow attacks at runtime has gone in several directions. We have identified six general categories or techniques, namely canary-based, boundary checking, copying and checking target data, encrypted instruction addresses, library wrappers, and non-executable and randomized memory. Our evaluation covers all but encrypted instruction addresses since we were not able to locate a publicly-available countermeasure performing such operations.

4.1 Canary-Based Tools

This technique was invented by Cowan *et al* [18] and prevents buffer overflows by adding a *canary value* to sensitive memory regions. The canary's integrity is checked before the sensitive memory is used. If the canary has been changed the sensitive memory may have been corrupted and the program is typically terminated. Other tools have adopted the canary, for instance detection of heap-based overflows targeting malloc linked lists by Robertson *et al* [36], Microsoft's `/GC` compiler flag [9], and stack protection with ProPolice by Etoh *et al* [21]. ProPolice is covered in our empirical evaluation and is presented in more detail in Sec. 5.1.

4.2 Boundary Checking Tools

Standard C and C++ do not have runtime bounds checking unlike newer languages such as Java and C#. This is one of the fundamental design decisions that make buffer overflow attacks possible. Researchers have implemented variants of C compilers that include boundary checking in binaries.

In 1997 Jones and Kelly presented a GCC compiler patch in which they implemented runtime bounds checking of variables [26]. Sadly their solution suffered from performance penalties of more than 400%, as well as incompatibility with real-world programs [17]. Ruwase and Lam continued Jones' and Kelly's work and have implemented a GCC patch called "CRED" [37]. CRED is covered in our empirical evaluation and is presented in section 5.5.

4.3 Tools Copying and Checking Target Data

StackShield [50] and Libverify [5] were the first buffer overflow prevention tools that used the technique of storing copies of return addresses on a separate stack. When a function returned, its stored return address is checked against the copy on the separate stack. If the addresses differed either the correct address was copied back or execution was halted. StackShield is a compiler patch whereas Libverify patched the code during load. Both StackShield and Libverify are covered in our empirical evaluation and are presented in section 5.2 and 5.3.

Chiueh and Hsu [13] presented a compiler patch called *RAD* in 2001. It used a separate stack to keep copies of return addresses similar to StackShield. Smirnov and Chiueh have continued the work and implemented a more complex GCC patch called *DIRA* [42]. Apart from the separate stack with copies of return addresses, DIRA keeps copies of function pointer values in a special buffer.Nebenzahl and Wool [30] have developed a technique for instrumenting Windows binaries at install-time with a separate stack for copies of return addresses.

4.4 Library Wrappers

Buffer overflow prevention through library wrappers was originally done by Baratloo, Singh, and Tsai, and their tool was called *Libsafe* [4]. It patches library functions in C that constitute potential buffer overflow vulnerabilities. In the patched functions a range check is made before the actual function call. As a boundary value Libsafe uses the old base pointer pushed onto the stack after the return address.

Avijit, Gupta and Gupta continued the work by Baratloo *et al* by implementing *LibsafePlus* and *TIED* [3, 2]. Their system collects and stores information about the sizes of both stack and heap buffers. This information is then used at runtime to ensure that no character buffers are written past their limit. Libsafe, LibsafePlus, and TIED are all covered in our empirical evaluation and are presented in more detail in Sections 5.3 and 5.4.

4.5 Non-Executable and Randomized Memory

The Linux kernel patch from the Openwall Project was the first to implement a non-executable stack [19]. Not allowing execution of code stored on the stack effectively stops execution of attack code injected on the stack and on the heap. In some cases, researchers have been able to circumvent this countermeasure by abusing certain traits of a program, e.g. convincing the Just-in-time compiler of ActionScript in Macromedia Flash to place attacker-code in their writable and executable memory pages [8].

Two more recent kernel patches that deny execution both on the stack and on the heap are PaX [23] and ExecShield [49]. They also randomize address offsets from the base of memory locations, called *Address Space Layout Randomization*, to further countermeasure buffer overflow attacks. DieHard [6] and its continuation DieHarder [31] are memory allocators which randomize the location of heap objects on the heap and require larger-than-needed address spaces to ensure probabilistic safety. Even though DieHard is publicly available we could not include it in our evaluation since RIPE is a process that attacks itself calculating the needed offsets from within its source code (see Sec. 8). This means, that RIPE would "unfairly" de-randomize DieHard and successfully perform all of the attacks.

5. EMPIRICAL EVALUATION SETUP

We have used RIPE to evaluate a number of preventive tools and techniques designed to counter buffer overflow attacks, namely ProPolice (canary-based), CRED (boundary checking), StackShield and Libverify (copying and checking target data), Libsafe, LibsafePlus, LibsafePlus+TIED (library wrappers), and PAE and XD (non-executable memory).

The theoretical number of attack forms produced by multiplying all the choices is 3,840 (4 locations * 16 target code pointers * 2 techniques * 3 variants of attack code without NOP sled variations * 10 functions being abused). However, that number incorporates 2,990 practically impossible attack forms. For instance it is not possible to perform a direct buffer overflow all the way from the BSS segment to the stack due to the unmapped heap pages and the guard page separating the stack and heap. Thus, the number of working attack forms is 850. In the empirical evaluation we have left out the three NOP versions and only executed with one (the simple NOP sled). Since none of the protection tools or techniques evaluated tries to detect NOP sleds *per se* including them would not change the results of the current analysis. Nevertheless RIPE supports three different NOP sled settings for attack code.

5.1 ProPolice

Hiroaki Etoh and Kunikazu Yoda from IBM Research in Tokyo have implemented a compiler protection called *ProPolice* [21]. It borrows the main idea from StackGuard—canary, or *guard* values to detect attacks on the stack. The guard is placed between the buffers and the old base pointer meaning it protects both the return pointer and the old base-pointer from direct overflows. In addition to the guard, ProPolice rearranges the local stack variables so that `char` buffers always are allocated at the bottom, next to the canary, where they cannot harm any other local variables if overflowed. Non-char buffer variables can only be attacked if they are part of a struct that also contains a buffer.

5.2 StackShield

StackShield is a compiler patch for GCC made by Vendicator [50]. In the current version 0.7 it implements three types of protection, two against overwriting of the return address (both can be used at the same time) and one against overwriting of function pointers.

5.2.1 Global Ret Stack

The *Global Ret Stack* protection of the return address is the default choice for StackShield. It is a separate stack for storing the return addresses of functions called during execution. The stack is a global array of 32-bit entries. Whenever a function call is made, the return address being pushed onto the normal stack is at the same time copied into the Global Ret Stack array. When the function returns, the return address on the normal stack is replaced by the copy on the Global Ret Stack. If an attacker had overwritten the return address in one way or another the attack would be stopped without terminating the process execution. Note that no comparison is made between the return address on the stack and the copy on the Global Ret Stack allowing the countermeasure to prevent but not to detect buffer overflows (and possible corruption of data due to them). The Global Ret Stack has by default 256 entries which limits the nesting depth to 256 protected function calls. Further function calls will be unprotected but execute normally.

5.2.2 Ret Range Check

A somewhat simpler but faster version of StackShield's protection of return addresses is the *Ret Range Check*. It uses a global variable to store the return address of the current function. Before returning, the return address on the stack is compared with the stored copy in the global variable. If there is a difference the execution is halted. Note that the Ret Range Check can detect an attack as opposed to the Global Ret Stack described above.

5.2.3 Protection of Function Pointers

StackShield also aims to protect function pointers from being overwritten. The idea is that function pointers normally should point into the text segment of the process' memory where the programmer is likely to have implemented the functions to point at. If the process can ensure that no function pointer is allowed to point into other parts of memory than the text segment, it will be impossible for an attacker to make it point at code injected into the process, since injection of data only can be done into the stack, the heap, the BSS, or the data segment.

StackShield adds checking code before all function calls that make use of function pointers. A global variable is then declared in the data segment and its address is used as a boundary value. The checking function ensures that any function pointer about to be dereferenced points to memory below the address of the global boundary variable. If it points above the boundary the process is terminated. This protection will give false positives if the program uses dynamically allocated function pointers.

5.3 Libsafe and Libverify

Another defense against buffer overflows presented by Arash Baratloo et al [4] is *Libsafe*. This tool actually provides a combination of static and dynamic intrusion prevention. Statically it patches library functions in C that constitute potential buffer overflow vulnerabilities. A range check is made before the actual function call which ensures that the return address and the base pointer cannot be overwritten. Further protection has been provided [5] with *Libverify* using a similar dynamic approach to StackGuard.

5.3.1 Libsafe

The key idea behind Libsafe is to estimate a safe boundary for buffers on the stack at run-time and then check this boundary before any vulnerable function is allowed to write to the buffer.

As a boundary value Libsafe uses the old base pointer pushed onto the stack after the return address. No local variable should be allowed to expand further down the stack than the beginning of the old base pointer. In this way a stack-based buffer overflow cannot overwrite the return address. This boundary is enforced by overloading `strcpy()`, `strcat()`, `getwd()`, `gets()`, `[vf]scanf()`, `realpath()`, and `[v]sprintf()` with wrapping functions. These wrappers first compute the length of the input as well as the allowed buffer size (i.e. from the buffer's starting point to the old base pointer) and then performs a boundary check. If the input is within the boundary the original functionality is carried out. If not the wrapper writes an alert to the system's log file and then halts the program. Observe that overflows within the local variables on the stack, such as function pointers, are not stopped.

5.3.2 Libverify

Libverify is an enhancement of Libsafe, implementing return address verification similar to StackShield. However, since this is a library it does not require recompilation of the software. As with Libsafe the library is pre-loaded and linked to any program running on the system. The key idea behind Libverify is to alter all functions in a process so that the first thing done in every function is to copy the return address onto a *canary stack* located on the heap, and the last thing done before returning is to verify the return address by comparing it with the address saved on the canary stack. If the return address is still correct the process is allowed to continue executing. However, if the return address does not match the saved copy, execution is halted and a security alert is raised. Libverify does not protect the integrity of the canary stack. They propose protecting it with `mprotect()` like *Return Address Defender*, RAD [13]. However, as in the RAD case this will most probably impose a serious performance penalty.

To be able to do this, Libverify has to transform the code of a given program. First each function is copied whole to the heap (requires executable heap) where it can be altered. Then the saving and verifying of the return address is injected into each function by overwriting the first instruction with a call to `wrapper_entry` and all return instructions with a call to `wrapper_exit`. The need for copying the code to the heap is due to the Intel CPU architecture. On other platforms this could be solved without copying the code [5]. Libverify is needed to give a more complete protection of the return address since Libsafe only addresses standard C library functions (as pointed out by Istvan Simon [41]). With Libsafe vulnerabilities could still occur where the programmer has implemented his/her own memory handling.

5.4 LibsafePlus and TIED

Avijit, Gupta and Gupta continued the work by Baratloo *et al* by implementing *LibsafePlus* and *TIED* [3, 2]. TIED collects static information and LibsafePlus collects dynamic information about the sizes of stack and heap buffers. This information is used runtime to ensure that no character buffers are written past their limit.

Static buffer sizes are collected compile-time by exploiting debugging information produced by a specific compiler option. Dynamic buffer size information is collected runtime by interception of calls to `malloc()` and `free()`. Finally, the original technique with wrappers for dynamically linked libraries handling strings is used to check the bounds. Their main contributions are a more precise boundary check of stack buffers than the previous solution, and a boundary check of heap buffers.

5.5 CRED

Ruwase and Lam continued Jones' and Kelly's boundary checking work and have implemented a GCC patch called "CRED", *C Range Error Detector* [37]. Their goals were for the runtime checks to impose less overhead and provide better compatibility. To enhance performance they only perform boundary checks on string buffers since they consider such buffers the most likely ones vulnerable to security attacks. With such a restriction most of the programs they tested suffered less than 26 % overhead. Worst case was a string-intensive email program which suffered 130 % overhead.

Compatibility was solved by storing out-of-bounds pointer values in so called *out-of-bounds objects*. If pointer arithmetics using the out-of-bounds pointer results in an in-bounds address the pointer is sanitized. All variables with a memory range such as arrays and structs get an associated *referent object* that keeps of pointer arithmetic and bounds. Pointer operations that reference memory outside the referent object are illegal. CRED allows out-of-bounds references to be part of arithmetic as long as the resulting access is within bounds.

5.6 Non-Executable Memory and Stack Protector (Ubuntu 9.10)

We have evaluated the buffer overflow prevention techniques used in Ubuntu 9.10 "Karmic" [33] which has several security features [34] relevant to buffer overflow prevention.

5.6.1 ASLR

Ubuntu 9.10 has five ASLR (Address Space Layout Randomization) features, four enabled by default—Stack ASLR, Libs/mmap ASLR, Exec ASLR, brk ASLR, and VDSO ASLR [34].

The fundamentals of defeating ASLR have been studied by Schacham *et al* [39]. Attackers may reduce the entropy present in a randomized address space by leaking information via format string attacks [20], buffer over-reads [45] or covering multiple bits of entropy per attack by using heap spraying, introduced by Hassell and Permeh [24]. RIPE does not use brute force, information leakage, or heap spraying to circumvent ASLR. While such attack methods are interesting, RIPE currently focusses on evaluating countermeasures performing attack detection or active prevention, not on countermeasures making attacks harder. RIPE calculates its offsets and target addresses at runtime and thus has information available *after* randomization. The effects of this decision are discussed in detail in Section 8.

5.6.2 Non-Executable Memory

Most modern CPUs protect against executing non-executable memory regions (heap, stack, etc), known either as Non-eXecute (NX) or eXecute-Disable (XD) [52]. This protection reduces the areas an attacker can use to perform arbitrary code execution. It requires that the kernel uses PAE, *Physical Address Extension*. Ubuntu 9.10, partially emulates this protection for processors lacking NX when running on a 32bit kernel. Our evaluation runs where performed on a machine with an Intel Core 2 Duo processor with XD support enabled.

5.6.3 Stack Protector (ProPolice)

Ubuntu 9.10 ships with a patched gcc using `-fstack-protector` by default. This protection is ProPolice, presented in section 5.1. RIPE was compiled with this gcc for the Ubuntu 9.10 evaluation.

6. EMPIRICAL EVALUATION RESULTS

In this section, we present the summary of our empirical evaluation of the protection tools and techniques presented in section 5. We then continue with detailed evaluation results for the top four. Full log files and test results will be published online when the study is presented. The summary of the empirical evaluation is presented in table 1. Our base-case is Ubuntu 6.06, a Linux distribution released in 2006 with no countermeasures against code-injection attacks.

6.1 Details for ProPolice

ProPolice is totally focused on protecting the stack and is successful in doing so for direct, stack-based buffer overflows except for structs with a buffer and function pointer. Also indirect, stack-based attacks are prevented because of the re-arranging of character buffers.

On the heap, BSS, and data segment ProPolice does not add any protective countermeasures so direct or indirect, heap/BSS/data-based attacks targeting any of the code pointers and abusing any of the functions will be successful. Indirect, heap/BSS/data-based attacks against longjmp buffers as stack variables or function parameters were not fully stable and thus categorized as partly successful.

6.2 Details for LibsafePlus + TIED

Libsafe's basic protection scheme is wrapping library functions (see list in section 5.3.1. This means that the only stable, successful attack forms were the ones abusing `memcpy()` and RIPE's "homebrew" memcpy equivalent since they are not wrapped.

Direct and indirect, stack/heap/BSS/data-based attacks targeting all code pointers are successful as long as they abuse `memcpy()` or RIPE's "homebrew" memcpy equivalent.

`snprintf()`, `sscanf()`, `strncpy()`, `strncat()`, `sscanf()`, `strcpy()`, `strcat()`, `fscanf()`, and `sprintf()` all were successfully abused a few times and therefore categorized as partly successful. Those partly successful attacks forms were spread across almost all other variables—direct and indirect, stack/BSS/data segment, injected code and return-into-libc and targeting return pointer, longjmp buffers, function pointers, old base pointer and structs.

6.3 Details for CRED

CRED fails to prevent direct and indirect, stack/BSS/data-based overflows toward function pointers, longjmp buffers, and structs for the library functions `sprintf()`, `snprintf()`, `sscanf()`, and `fscanf()`. The attacks against structs are also successful for `memcpy()` and homebrew memcpy equivalent and are the only attacks successful from buffers on the heap. The exception to the above is indirect attacks from the BSS and data segments targeting a longjmp buffer as stack variable. There was some instability in those attacks and therefore they were categorized as partly successful.

6.4 Details for Non-Executable Memory and Stack Protector (Ubuntu 9.10)

Ubuntu 9.10 with non-executable memory and stack protection scored the best in our evaluation. All attack forms that involved the injection of new code in a process' address space failed, due to the policy that a memory page can be either writable, or executable but not both. Also, any attack against the return address of a function was blocked due to the presence of a canary and the re-ordering of variables done by ProPolice. Strackx et. al [45] have shown cases where an attack against the stack is possible even in the presence of canary-based countermeasures, however we decided not to include such an attack in the current version.

All struct attack forms were successful meaning all locations and all abused functions worked, verifying the limitations of ProPolice. Additionally all direct attacks against function pointers on the heap and the data segment were successful. Indirect attacks against the old base pointer works in general on the heap, BSS, and data segment for `memcpy()`, `strcpy()`, `strncpy()`, `sprintf()`, `snprintf()`, `strcat()`, `strncat()`, `sscanf()`, `fscanf()`, and homebrew memcpy equivalent. But there were some instability for 10 of those combinations.

6.5 A Note on Evaluation of StackShield

The testbed execution for StackShield strangely claims only 1810 impossible attack forms whereas all the others say 2990. We figure this is because of StackShield's transformations and have manually removed the missing 1180 impossible attack forms from StackShield's failed attacks since successful and partly successful attacks are obviously possible. If in fact StackShield's transformation makes 1180 extra attack forms possible, that means an increased attack surface and not enhanced protection.

6.6 Potential Shortcomings

6.6.1 Synthesized vs Real-World Code Testbeds

RIPE is a synthesized testbed, deliberately vulnerable, and a program with the sole purpose of conducting attacks against itself and recording their success or failure. Compared to real-world code testbeds, RIPE offers no evaluation of scalability, complexity, or performance. We see merits in both approaches—the main one for synthesized testbeds being the possibility to enumerate and combine attack forms to provide good coverage.

Setup	**Overall effectiveness**	Successful attacks	Partly successful attacks	Failed attacks
Ubuntu 6.06 (no protection)	**0%**	838 (99%)	12 (1%)	0 (0%)
Libsafe (lib wrapper)	**7%**	777 (91%)	16 (2%)	57 (7%)
LibsafePlus (lib wrapper)	**19%**	669 (79%)	16 (2%)	165 (19%)
StackShield (copy & check)	**36%**	533 (63%)	7 (1%)	310 (36%)
ProPolice (canary-based)	**40%**	501 (59%)	9 (1%)	340 (40%)
LibsafePlus+TIED (lib wrapper)	**77%**	170 (20%)	23 (3%)	657 (77%)
CRED (boundary checking)	**79%**	172 (20%)	4 (0.5%)	674 (79%)
Non-exec + stack prot (Ubuntu 9.10)	**89%**	80 (9%)	10 (1%)	760 (89%)

Table 1: Summary of empirical evaluation results using RIPE's 850 buffer overflow attack forms. Overall effectiveness is the percentage of attack forms prevented. Successful attacks give repeatable arbitrary code execution. Partly successful attacks are sometimes successful, sometimes not and in general less stable. Failed attacks are repeatably prevented.

6.6.2 False Negatives and Result Manipulation

The kind of evaluation RIPE provides is susceptible to both false negatives and manipulation. An evaluated tool can prevent RIPE's implementation of a given attack form but still allow for exploitations of such attack forms in general. RIPE only provides one vulnerability and matching payload for each of the 850 attack forms, whereas in theory there are infinite variations of both. Such a case could be interpreted as a false negative. Therefore, evaluation results should be interpreted as an upper bound on the preventive effectiveness for the RIPE attack forms—there might be further successful attack forms among the 850.

Further more, researchers could inspect or observe the specifics of how RIPE implements certain attack forms and adjust their countermeasures to prevent exactly those attacks. While this could be based on bad intentions and effectively be result manipulation, it doesn't have to be. Such RIPE-specific prevention might evolve over time when fine tuning to give good evaluation results. Therefore, care has to be taken when comparing RIPE evaluation outcomes between countermeasures. We believe that it is in every researcher's own interest to use RIPE to evaluate fairly and since RIPE is free software any necessary testbed augments can be implemented and published.

7. RELATED WORK

Pincus and Baker present an overview of recent advances in exploitation of buffer overflows [32]. Their main conclusion is that often heard assumptions about buffer overflows are not true—buffer overflows do not all inject code, do not all target the return address, and do not all abuse buffers on the stack. The article briefly discusses (1) injection of attack parameters instead of attack code, (2) attacks targeting function pointers, data pointers, exception handlers, and pointers to virtual function tables in C++, and (3) heap-based overflows.

Michael Zhivich *et al* published "Dynamic Buffer Overflow Detection" in 2005 [54]. They use a collection of 55 small, synthesized C programs that contain buffer overflows to evaluate. They have several "dimensions" in their testbed. They differentiate between *discrete* overflows of up to 8 bytes of memory, and *continuous* overflows resulting from multiple consecutive writes. They have several buffer types—`char`, `int`, `float`, `func *`, and `char *` and they are spread in the same four memory locations as we have; stack, heap, BSS, and data segment. They have buffers in struct, array, union, and array of structs. Lib functions they abuse are `(f)gets`, `(fs)scanf`, `fread`, `fwrite`, `sprintf`, `str(n)cpy`, `str(n)cat`, and `memcpy`. An attack is judged as prevented if it's detected or if a segmentation fault occurs. The top performing tools in their study are Insure++, CCured and CRED. They also evaluate the tools against approximately 100 line models of fourteen historic vulnerabilities in bind, sendmail, and wu-ftpd. CCured, CRED, and TinyCC came out on top, detecting about 90% of the overflows. Unfortunately their testbed is not available which means their study cannot be repeated and their test cases cannot be used for future evaluations. Also they do not try all possible attack combinations nor publish exactly which buffer overflows worked and which didn't. In contrast, RIPE is meant to be a publicly available evaluator which researchers can use to report and compare the coverage of their security mechanisms against a large but well-defined set of real-world attacks.

8. FUTURE WORK

As mentioned in previous sections, RIPE is a process that attacks itself and then checks the success or failure of the launched attacks. Due to the fact that the attack code is part of the vulnerable process, any countermeasures relying on the secrecy of memory locations are defeated since RIPE has access to the addresses of both the overflowing buffer and the target. RIPE could be extended with a *save/load offsets* feature allowing offsets from one execution to be used in a subsequent run. This would allow the evaluation of countermeasures that rely on memory randomization such as ASLR or DieHard [6] and DieHarder [31]. Heap spraying and information leakage attacks could also be added to "assist" an attacker in de-randomizing certain countermeasures. We are also currently considering the addition of non-control data attacks [12] which would allow for evaluation of countermeasures such as *data space randomization*[7] and ValueGuard [48].

9. CONCLUSIONS

Even though hundreds of papers have been published on the problem of buffer overflows and code injection attacks, modern software still is plagued by improper checking of user input attesting to the fact that this is still an open research problem. In this paper we presented RIPE, a Runtime Intrusion Prevention Evaluator which executes a total of 850 buffer-overflow attacks against popular defense mechanisms. The main purpose of RIPE is to provide a freely available testbed which developers of defense mechanisms can use to quantify the security coverage of their proposals and compare themselves against previous work using a well-defined and real-world set of attacks.

Acknowledgements:.

This research is partially funded by the national computer graduate school in computer science (CUGS) commissioned by the Swedish government, the board of education and the Interuniversity Attraction Poles Programme Belgian State, Belgian Science Policy, the IBBT and the Research Fund K.U.Leuven.

Initial parts of the testbed extensions were built by Pontus Viking as part of his Master's Thesis [51].

We are grateful to the readers who have previewed and improved our paper, especially Martin Johns.

10. AVAILABILITY

RIPE is free software released under the MIT licence and available on GitHub: `https://github.com/johnwilander/RIPE`

11. REFERENCES

[1] Akritidis, P., Markatos, E., Polychronakis, M., and Anagnostakis, K. Stride: Polymorphic sled detection through instruction sequence analysis. In *Security and Privacy in the Age of Ubiquitous Computing*, R. Sasaki, S. Qing, E. Okamoto, and H. Yoshiura, Eds., vol. 181 of *IFIP Advances in Information and Communication Technology*. Springer Boston, 2005, pp. 375–391.

[2] Avijit, K., Gupta, P., and Gupta, D. Tied, libsafeplus: Tools for runtime buffer overflow protection. In *Proceedings of The 13th USENIX Security Symposium* (San Diego, USA, August 2004), pp. 45–56.

[3] Avijit, K., Gupta, P., and Gupta, D. Binary rewriting and call interception for efficient runtime protection against buffer overflows: Research articles. *Softw. Pract. Exper. 36* (July 2006), 971–998.

[4] Baratloo, A., Singh, N., and Tsai, T. Libsafe: Protecting critical elements of stacks. White Paper `http://www.research.avayalabs.com/project/libsafe/`, December 1999.

[5] Baratloo, A., Singh, N., and Tsai, T. Transparent run-time defense against stack smashing attacks. In *Proceedings of the 2000 USENIX Technical Conference* (San Diego, California, USA, June 2000).

[6] Berger, E. D., and Zorn, B. G. Diehard: probabilistic memory safety for unsafe languages. In *Proceedings of the 2006 conference on Programming language design and implementation* (Ottawa, ON, 2006), ACM Press, pp. 158–168.

[7] Bhatkar, S., and Sekar, R. Data space randomization. In *Proceedings of the 5th international conference on Detection of Intrusions and Malware, and Vulnerability Assessment (DIMVA '08)* (July 2008).

[8] Blazakis, D. Interpreter Exploitation: Pointer Inference and JIT Spraying. In *BlackHat DC* (2010).

[9] Bray, B. Compiler security checks in depth. `http://msdn.microsoft.com/en-us/library/aa290051(v=vs.71).aspx`, February 2002.

[10] Bulba, and Kil3r. Bypassing StackGuard and StackShield. Phrack Magazine Volume 10, Issue 56 `http://www.phrack.org/phrack/56/p56-0x05`, May 2000.

[11] Castro, M. Securing software by enforcing data-flow integrity. In *Proceedings of the 7th Symposium on Operating Systems Design and Implementation* (2006), pp. 147–160.

[12] Chen, S., Xu, J., Sezer, E. C., Gauriar, P., and Iyer, R. K. Non-control-data attacks are realistic threats. In *14th USENIX Security Symposium* (2005).

[13] cker Chiueh, T., and Hsu, F.-H. RAD: A compile-time solution to buffer overflow attacks. In *Proceedings of the 21th International Conference on Distributed Computing Systems (ICDCS)* (Phoenix, Arizona, USA, April 2001).

[14] Clause, J., Li, W., and Orso, R. Dytan: A generic dynamic taint analysis framework. In *Proceedings of the International Symposium on Software Testing and Analysis* (2007), pp. 196–206.

[15] Costa, M., Crowcroft, J., Castro, M., and Rowstron, A. Can we contain internet worms? In *Proceedings of Third Workshop on Hot Topics in Networks, HotNets-III* (San Diego, CA USA, November 2004).

[16] Costa, M., Crowcroft, J., Castro, M., Rowstron, A., Zhou, L., Zhang, L., and Barham, P. Vigilante: End-to-end containment of internet worms. In *Proceedings of the Symposium on Systems and Operating Systems Principles (SOSP)* (2005), pp. 133–147.

[17] Cowan, C., Beattie, S., Day, R. F., Pu, C., Wagle, P., and Walthinsen, E. Protecting systems from stack smashing attacks with StackGuard. Linux Expo `http://www.cse.ogi.edu/~crispin/`, May 1999.

[18] Cowan, C., Pu, C., Maier, D., Walpole, J., Bakke, P., Beattie, S., Grier, A., Wagle, P., Zhang, Q., and Hinton, H. StackGuard: Automatic adaptive detection and prevention of buffer-overflow attacks. In *Proceedings of the 7th USENIX Security Conference* (San Antonio, Texas, January 1998), pp. 63–78.

[19] Designer, S. Linux kernel patch from the openwall project. `http://www.openwall.com/linux/README`.

[20] Durden, T. Bypassing pax aslr protection. `http://www.phrack.com/issues.html?issue=59&id=9`, July 2002.

[21] Etoh, H. GCC extension for protecting applications from stack-smashing attacks. `http://www.trl.ibm.com/projects/security/ssp/`, June 2000.

[22] Gajdos, P., and Kornacker, C. Cve-2009-4035 xpdf: buffer overflow in fofitype1. `https://bugzilla.redhat.com/show_bug.cgi?id=541614`, December 2009.

[23] grsecurity. Pax. `http://pax.grsecurity.net/`.

[24] Hassell, R., and Permeh, R. Microsoft internet information services remote buffer overflow. http://www.eeye.com/Resources/Security-Center/Research/Security-Advisories/AD20010618, 6 2001.

[25] Howard, M. Evils of strncat and strncpy - answers. http://blogs.msdn.com/b/michael_howard/archive/2004/12/10/279639.aspx, December 2004.

[26] Jones, R., and Kelly, P. Backwards-compatible bounds checking for arrays and pointers in C programs. In *Proceedings of the Third International Workshop on Automatic Debugging AADEBUG'97* (1997).

[27] Kc, G. S., Keromytis, A. D., and Prevelakis, V. Countering code-injection attacks with instruction-set randomization. In *Proceedings of The 10th ACM Conference on Computer and Communications Security* (Washington D.C., USA, 2003), pp. 272–280.

[28] Moore, D., Paxson, V., Savage, S., Shannon, C., Staniford, S., and Weaver, N. Inside the slammer worm. *IEEE Security and Privacy 1*, 4 (2003), 33–39.

[29] Moore, D., Shannon, C., and claffy, k. Code-red: a case study on the spread and victims of an internet worm. In *2nd ACM Workshop on Internet measurment* (2002).

[30] Nebenzahl, D., and Wool, A. Install-time vaccination of windows executables to defend against stacksmashing attacks. In *Proceedings of The 19th IFIP International Information Security Conference* (2004).

[31] Novark, G., and Berger, E. D. Dieharder: securing the heap. In *Proceedings of the 17th ACM conference on Computer and communications security* (New York, NY, USA, 2010), CCS '10, ACM, pp. 573–584.

[32] Pincus, J., and Baker, B. Beyond stack smashing: Recent advances in exploiting buffer overruns. *IEEE Security and Privacy 2*, 4 (2004), 20–27.

[33] Project, T. U. Ubuntu 9.10 karmic. http://releases.ubuntu.com/karmic/.

[34] Project, T. U. Ubuntu security feature matrix. https://wiki.ubuntu.com/Security/Features.

[35] Qin, F., Wang, C., Li, Z., seop Kim, H., Zhou, Y., and Wu, Y. Lift: A low-overhead practical information flow tracking system for detecting security attacks. *Microarchitecture, IEEE/ACM International Symposium on 0* (2006), 135–148.

[36] Robertson, W., Kruegel, C., Mutz, D., and Valeur, F. Run-time detection of heap-based overflows. In *Proceedings of The 17th Large Installation Systems Administration Conference* (San Diego, USA, October 2003).

[37] Ruwase, O., and Lam, M. S. A practical dynamic buffer overflow detector. In *Proceedings of The 11th Annual Network and Distributed System Security Symposium* (San Diego, USA, February 2004).

[38] Shacham, H. The geometry of innocent flesh on the bone: Return-into-libc without function calls (on the x86). In *Proceedings of CCS 2007* (Oct. 2007), S. De Capitani di Vimercati and P. Syverson, Eds., ACM Press, pp. 552–61.

[39] Shacham, H., Page, M., Pfaff, B., Goh, E.-J., Modadugu, N., and Boneh, D. On the effectiveness of address-space randomization. In *Proceedings of the 11th ACM conference on Computer and communications security, CCS '04* (2004).

[40] Sidiroglou, S., and Keromytis, A. D. Countering network worms through automatic patch generation. *Security & Privacy, IEEE 3*, 6 (November–December 2005), 41–49.

[41] Simon, I. A comparative analysis of methods of defense against buffer overflow attacks. http://www.mcs.csuhayward.edu/~simon/security/boflo.html, January 2001.

[42] Smirnov, A., and cker Chiueh, T. Dira: Automatic detection, identification, and repair of control-hijacking attacks. In *Proceedings of The 12th Annual Network and Distributed System Security Symposium* (2005).

[43] Spafford, E. H., and Spafford, E. H. The internet worm program: An analysis. *Computer Communication Review 19* (1988).

[44] SPEC - Standard Performance Evaluation Corporation. http://www.spec.org/.

[45] Strackx, R., Younan, Y., Philippaerts, P., Piessens, F., Lachmund, S., and Walter, T. Breaking the memory secrecy assumption. In *2nd European Workshop on System Security* (2009).

[46] Tuck, N., Calder, B., and Varghese, G. Hardware and binary modification support for code pointer protection from buffer overflow. In *Proceedings of the 37th Intl Symposium on Microarchitecture, MICRO 04* (2004), pp. 209–220.

[47] US-CERT. Vulnerability notes database. http://www.kb.cert.org/vuls.

[48] Van Acker, S., Nikiforakis, N., Philippaerts, P., Younan, Y., and Piessens, F. Valueguard: Protection of native applications against data-only buffer overflows. In *Proceedings of the Sixth International Conference on Information Systems Security (ICISS)* (2010).

[49] van de Ven, A., and Molnar, I. Execshield. http://people.redhat.com/mingo/exec-shield/docs/WHP0006US_Execshield.pdf.

[50] Vendicator. Stack Shield technical info file v0.7. http://www.angelfire.com/sk/stackshield/, January 2001.

[51] Viking, P. Comparison of dynamic buffer overflow protection tools. Master's thesis, Linkopings universitet, February 2006.

[52] Wikipedia. Wikipedia, nx bit. http://en.wikipedia.org/wiki/NX_bit.

[53] Wilander, J., and Kamkar, M. A comparative study of publicly available tools for dynamic buffer overflow prevention. In *Proceedings of the 10th Network & Distributed System Security Symposium* (San Diego, California, February 2003).

[54] Zhivich, M., Leek, T., and Lippmann, R. Dynamic buffer overflow detection. Workshop on the Evaluation of Software Defect Detection Tools, co-located with PLDI 2005 http://ewww.cs.umd.edu/~pugh/BugWorkshop05/, June 2005.

Hit ’em Where it Hurts: A Live Security Exercise on Cyber Situational Awareness

Adam Doupé, Manuel Egele, Benjamin Caillat, Gianluca Stringhini, Gorkem Yakin, Ali Zand, Ludovico Cavedon, and Giovanni Vigna
University of California, Santa Barbara
{adoupe, maeg, benjamin, gianluca, gyakin, zand, cavedon, vigna}@cs.ucsb.edu

ABSTRACT

Live security exercises are a powerful educational tool to motivate students to excel and foster research and development of novel security solutions. Our insight is to design a live security exercise to provide interesting datasets in a specific area of security research. In this paper we validated this insight, and we present the design of a novel kind of live security competition centered on the concept of Cyber Situational Awareness. The competition was carried out in December 2010, and involved 72 teams (900 students) spread across 16 countries, making it the largest educational live security exercise ever performed. We present both the innovative design of this competition and the novel dataset we collected. In addition, we define Cyber Situational Awareness metrics to characterize the toxicity and effectiveness of the attacks performed by the participants with respect to the missions carried out by the targets of the attack.

1. INTRODUCTION

In recent years, security attacks have become increasingly wide-spread and sophisticated. These attacks are made possible by vulnerable software, poorly configured systems, and a lack of security awareness and education of end users.

While a large portion of the security research efforts are focused on developing novel mechanisms and policies to detect, block, and/or prevent security attacks, there is also the need for the development of novel approaches to educate those who create the computer infrastructure, as well as those who use it everyday.

This is an often-overlooked aspect of computer security, but a critical one. Almost all sophisticated, widely deployed, security mechanisms can be made useless by luring an unsuspecting user (or a developer) into performing actions that, eventually, will compromise the security of their environment. A clear example of the popularity of these attacks is the proliferation of fake anti-virus scams, in which users who are not technically savvy are conned into installing a Trojan application [17].

Security education can be performed at different levels to reach different segments, from everyday Internet users, to high school students, to undergraduate and graduate students. Recently, competition-based educational tools have become popular in graduate and undergraduate education, as competition among students fosters creativity, innovation, and the desire to excel.

Previous work has described traditional “capture the flag competitions” [19, 20], and, more recently, new designs for this type of competition [2]. The development of new designs improved the competition and forced the participants to analyze and understand unfamiliar, complex sets of interdependent components, similar to those that are part of real-life networks and malware infrastructures.

Our novel insight is that these competitions, can, in addition to their educational value, provide interesting datasets that can be used in research. To validate this idea we designed and developed a novel security competition based on the concept of *Cyber Situational Awareness* (described in Section 2). The competition is called the iCTF (international Capture the Flag) and was carried out on December 3rd, 2010, involving 72 teams and 900 students, making it the largest live educational security exercise ever performed.

This paper presents the design of the competition, the data that was collected, and the lessons learned. The data is the first publicly available dataset that explicitly supports research in Cyber Situational Awareness.

In summary, this paper adds the following contributions:

- We describe the design and implementation of a novel computer security competition, whose goal is to not just foster computer security education, but to create a Cyber Situational Awareness dataset.
- We analyze the collected dataset and discuss its use in Cyber Situational Awareness research, introducing a novel metric that characterizes the effectiveness of attacks with respect to a specific mission.
- We discuss the lessons learned from the competition, and we provide suggestions to other educators that might implement similar competitions.

2. BACKGROUND AND HISTORY

In this section, we provide background on two of the most important aspects of this paper: the design and execution of live security competitions, and the concepts associated with Cyber Situational Awareness.

2.1 Live Security Competitions

Security challenges have been a way to attract the interest of security researchers, practitioners, and students. Live security challenges add a real-time factor that supports deeper involvement and introduces the "crisis factor" associated with many real-life security problems: "*something bad is happening right now and has to be taken care of.*"

There have been a number of live security challenges, but the best-known competition is DefCon's Capture The Flag (CTF). This competition started with a simple design, where a host with vulnerable services was made available to the participants, who would attack the target concurrently. Whoever was able to break a service and steal the flag first, obtained the points associated with that service. The original design was changed in 2002. In this edition of DefCon's CTF, the participating teams received an identical copy of a virtualized system containing a number of vulnerable services. Each team ran their virtual machine on a virtual private network (VPN), with the goal of maintaining the service's availability and integrity whilst concurrently compromising the other teams' services. Since each team had exactly the same copy of the services, the participants had to analyze the services, find the vulnerabilities, patch their own copies, and break into the other teams' services and steal the associated flags. Every other DefCon CTF following 2002 used more or less the same design [4].

Even though DefCon's CTF was designed to test the skills of hackers and security professionals, it was clear that the same type of competition could be used as an educational tool. One of the major differences between the iCTF and DefCon's CTF is that the iCTF involves educational institutions spread out across the world, where the DefCon CTF allows only locally-connected teams. Therefore, DefCon requires the physical co-location of the contestants thus constraining participation to a limited number of teams. By providing remote access, the iCTF allows dozens of remotely located teams to compete.

The iCTF editions from 2003 to 2007 were similar to the DefCon CTF: the participants had to protect and attack a virtualized system containing vulnerable services [20]. In 2008 and 2009, two new designs were introduced: in 2008, the competition was designed as a "treasure hunt," where the participants had to sequentially break into a series of hosts; in 2009, the competition focused on drive-by-download attacks, and the targets were a large pool of vulnerable browsers [2]. The iCTF inspired other educational hacking competitions, e.g., CIPHER [12] and RuCTF [18].

Recently, a different type of competition has received a significant amount of attention. In the Pwn2Own hacking challenge [13] participants try to compromise the security of various up-to-date computer devices such as laptops and smart phones. Whoever successfully compromises a device, wins the device itself as a prize. This competition is solely focused on attack, does not have an educational focus, and does not allow any real interaction amongst the participants who attack a single target in parallel.

Another interesting competition is the Cyber Defense Exercise (CDX) [1,10,14], in which a number of military schools compete in protecting their networks from external attackers. This competition differs from the UCSB iCTF in a number of ways. First, the competition's sole focus is on defense. Second, the competition is scored in person by human evaluators who observe the activity of the participants, and score them according to their ability to react to attacks. This evaluation method is subjective and requires a human judge for each team thus rendering it impractical in a large-scale on-line security competition.

The 2010 iCTF differed from the 2009 iCTF [2] in the following way: we realized that a live security exercise could be structured to create a dataset to enable security research. We utilized this idea in the 2010 iCTF by creating a Cyber Situational Awareness security competition that would generate a useful Cyber Situational Awareness dataset.

2.2 Cyber Situational Awareness

Cyber Situational Awareness (CSA) is an extension of traditional Situational Awareness (SA) to computer networks. The idea behind SA is that by analyzing the surrounding environment and putting perceived events into the context of the current mission, it is possible to improve decision-making. In the cyber-world, the concept of Situational Awareness includes the concept of mission awareness, which is the analysis of network events with respect to the mission being carried out by a particular organization.

One of the most important ideas behind CSA is that not all attacks have the same impact. The relevance of an attack is determined by the importance of the target with respect to a specific mission and a specific moment in time. For example, an attack against an FTP server could be harmless if the server is not a necessary component for the currently executing mission(s) in the next, say, eight hours, because within that time frame the server could be fixed/cleaned and it could be available when needed. Instead, consider an attack against a VoIP router when a strategic meeting must use that particular piece of infrastructure. The attack will directly impact the mission being carried out, and might impose delays or cause the mission to fail.

There are several challenges in CSA. First of all, it is difficult to correctly model missions. In many cases, organizations and companies are not even aware of their cyber-missions. Usually, identifying cyber-missions is easier in environments where repetitive tasks are performed cyclically. For example, banks have well-defined missions with tasks that must be carried out in specific sequences (e.g., closing balances, reconcile balance sheets) and must be performed within a certain time limit (e.g., midnight of the current day). Another example is military systems, where cycles of observation/analysis/operation phases are carefully followed, with precise time frames and clear dependencies.

In all these cases, one must choose a particular format to precisely describe a mission. A simple solution is to use Gantt charts [3], which clearly represent the duration and dependency of different tasks. For the cyber-missions described in this paper, we used *Missionary*, a Petri net [11] based formalism we created which extends the basic Petri net model with timing and code fragments associated with transitions and states. In this formalism, the tasks are represented by the states of the Petri net. A token in a state characterizes an active instance of the task. A task terminates when a token is removed from the corresponding state as a side-effect of the firing of a transition. Analogously, a task starts when a token is created in a state as the side-effect of the firing of a transition. Peterson [11] has a detailed description of Petri nets and their extensions.

Another challenge in CSA is to represent the dependency between cyber-missions and both the *human actors* and *as-*

sets involved in the missions [5]. For the sake of simplicity we do not address the former. For the latter, we used Missionary's service composition formalism, which allows the association of different types of service compositions to a task in a mission. In a nutshell, the formalism allowed us to specify services that were associated with a state in the Petri net, thus creating an association between a mission task and the services necessary to carry out the task.

3. 2010 iCTF

The iCTF competition was held on December 3rd, 2010, and lasted from 09:00 until 17:00, PST.

3.1 Pre-competition setup

Registration for the iCTF competition began a month before the start date. Attempting to alleviate the VPN connection problems that can occur on the day of the competition, we distributed a VMware [21] image along with VPN connection instructions to each team 11 days before the competition. The VMware image was meant as an example of the type of VMware image that would be used for the competition. We took particular care in making sure that the teams solved their connectivity problems well in advance, so that they could focus on the competition.

3.2 Story

The theme of the iCTF competition was "Mission awareness in state-sponsored cyberwar." The following text was given to the teams the day before the competition:

> *The country of Litya has become a major center for illegal activities of all kinds. The country is ruled by the ruthless dictator Lisvoy Bironulesk, who has pioneered the use of large malware infrastructures in order to support Litya's economy. Recently, he has claimed that Litya has "a botnet in every country."*
>
> *His complete disregard for international laws, his support of banking fraud and phishing scams, together with his well-known taste for underage girls has finally brought the attention of the international community into his shady dealings.*
>
> *Enough is enough. Now, the affected nations have decided to strike back. Spies who infiltrated Litya's corrupt administration have leaked plans of the most critical missions carried out in the country. These plans appear to describe the various activities of each mission, their ordering and timing, and their dependency on particular services.*

In this scenario, each team represented a country with the common goal of dismantling Litya's infrastructure, thus ending Bironulesk's reign. In addition to this text, the teams were given a number of images that described the various "missions" carried out by Litya. One of the missions is shown in Figure 1.

3.3 Competition Description

At a high level, the competition was designed to force the teams to exploit services at specific times, when they are most needed by Litya, thus emulating a Cyber Situational Awareness scenario. The teams had to access the services, first by bribing Litya's administrators, then by keeping their VMware image connected to a "mothership." If the teams generated an intrusion detection system alert, they were blocked from the network for a fixed amount of time.

3.3.1 Scoring

There were two types of scores: money and points. The team with the highest points won the competition, thus points were more important than money. Points were acquired by exploiting services at the correct time. However, if a team did not have any money, they would be shut off from the network and not be able to score any points. In addition to starting the competition with 1000 in money and zero points, each team earned money by solving challenges.

3.3.2 Firewall and IDS

A substantial innovation introduced in the iCTF was creating an intrusion prevention system (IPS) by connecting the Snort [16] intrusion detection system to the firewall. If Snort detected an intrusion attempt (alert) from a team on traffic directed towards Litya's services, the offending team was shut off from the network for ten minutes. The team would either have to wait until connectivity was allowed again or spend money bribing Litya's network administrators to gain access to the network for a certain amount of time. The teams had full knowledge of the Snort version and configuration, thus, they could predict if their traffic would generate an alert. Connecting Snort to the firewall forced the teams to come up with novel ways to circumvent an IDS.

3.3.3 Botnet

Bribing Litya's network administrators for access opened up the network for a limited amount of time (proportional to the amount of money used to bribe). To remain connected to the network, the teams needed to run a bot, which we provided 2 hours before the competition. This bot would connect to a mothership every 30 seconds and while the bot was connected to the mothership it would drain money from the team at a rate of 6 money per minute. As long as the bot remained connected to the mothership, the team had money, *and* the team didn't generate any Snort alerts, they could access the services. The two means of connecting to the network (bot connection or bribing) forced teams to make strategic decisions about when to connect, when to attack, how to attack, and when to bribe (spend money). These strategic decisions added another dimension to the iCTF competition: Teams had to decide the proper allocation of money (bot connection or bribing) to maximize their access to the network and thus maximize points.

Like a real-world *bot*, these machines were "compromised" and had 3vilSh3ll [15], a backdoor bind connect, running on port 8000. This allowed anyone to connect on port 8000, supply the password: `hacked`, and obtain a root shell. The idea was to encourage teams to be careful about their firewall, and force them to defensively select what traffic they allowed into their network.

3.3.4 Challenges

To gain money to bribe the Litya network administrators, as well as allow the mothership to steal money and remain connected to the network, teams needed to solve challenges. We created 33 challenges to provide multiple ways to earn money, but also to offer opportunities to test and improve

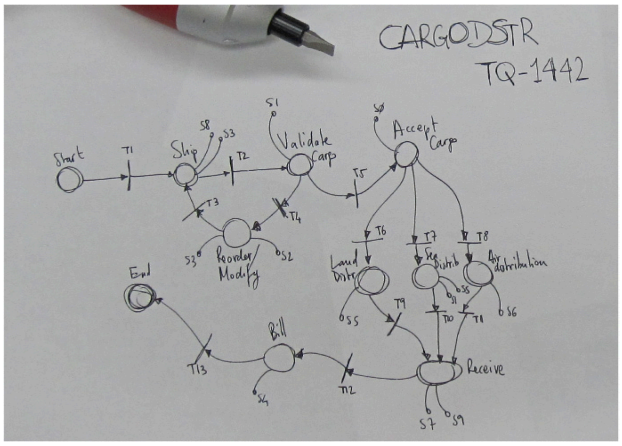


Figure 1: CARGODSTR mission that was distributed to the teams.

different skills, from cryptanalysis to forensics, program and network analysis.

3.3.5 *Scoreboard*

In a capture the flag competition, a scoreboard showcasing the current status and ranking of each team is vital. For the iCTF competition, we also needed to show the connection status of each team; if they were connected to the network, and why they were disconnected: Either from an IDS alert, lack of a bot connection, or lack of money. The scoreboard also showed the history of each team's money and points. The scoreboard is a very important piece of the infrastructure, because it provides immediate feedback to the teams about the success (or failure) of their attacks. Unfortunately, we had some glitches in our scoreboard that we will discuss in Section 5.

3.3.6 *Missions*

The day before the competition each team received an email containing a link to four pictures. Each picture contained a description of a Cyber Situational Awareness mission, in the form of a hand-drawn Petri net. Figure 1 shows one of these missions, the CARGODSTR mission. In the Petri net, all of the transitions were named (although not unique across the missions and even within some missions), as were most of the states. Some of the states were associated with one or more of the 10 services (S0-S9). For example, the "Receive" state in the lower right of Figure 1 is associated with services S7 and S9. The four Petri net missions given to the teams are graphically shown in Figure 2.

A service that we ran executed the Petri nets by inserting a token in each of the "Start" states, and running each Petri net separately. At each time-step, for each of the missions, one of the eligible transitions (a transition where all inputs had tokens) was randomly chosen to fire. Then, the token was consumed on all the inputs to the chosen transitions, and a token was placed on all the outputs. The four chosen transitions (one from each mission) were leaked to the teams after each time-step. Then, after each mission was executed, the service suspended for a random amount of time between one and two minutes, and the process repeated until the end of the competition. If a token was in an "End" state, or the network became stuck (no eligible transitions) then the network was reset.

Service	Vulnerability
LityaBook	Cross-Site Scripting
LityaHot	Session Fixation
icbmd	Foam Rocket Firing
StormLog	Off-By-One Overflow
StolenCC	Perl's `open` abuse
SecureJava	Broken Crypto
IdreamOfJeannie	Java JNI Off-By-One Error
WeirdTCP	TCP IP Spoofing
MostWanted	SQL-Injection
OvertCovert	Format String

Table 1: Brief description of vulnerable services.

For example, running the CARGODSTR mission, Figure 1 and Figure 2a, would involve first placing a token in its "Start" position. As `T1` is the only eligible transition to fire, it is chosen, and this information is leaked to the teams. The token moves from "Start" to "Ship." Because services `S8` and `S3` are associated with the "Ship" state, and it has a token, they are active. After a pause of one to two minutes, the next time-step occurs. Once again, there is only one eligible transition, `T2`. It is chosen, and the token on "Ship" moves to "Validate Cargo." Now, services `S8` and `S3` are no longer active, but service `S1` becomes active. After another pause, the process repeats, but with two eligible transitions, `T4` and `T5`. One of these is randomly chosen, say `T5`, and the token moves. This process repeats until the end of the competition. A visualization of the execution of all four missions throughout the iCTF competition is available[1].

From this example of the execution of the CARGODSTR mission, the teams received the sequence: `T1 T2 T5`. With only this information, they had to reverse engineer the state of the mission to find out which services were active. The teams would then attack only the active services. In the CARGODSTR mission, this is simple because the transitions are unique, however this is not the case for all the missions, as shown in the SEDAFER mission (Figure 2d).

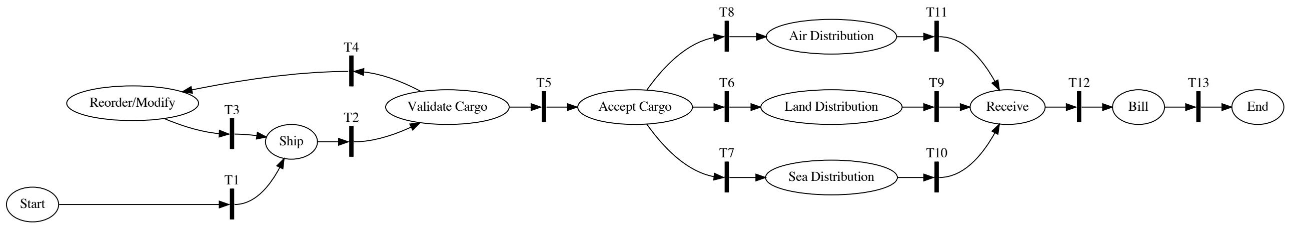
3.3.7 *Flags*

A flag was a sequence of hexadecimal values prefixed with `FLG` and was specific to a service. Flags were not directly accessible: a service must be compromised to access the associated flag. Therefore, flags are used by the participants as proof that, at a certain time, they were able to compromise a specific service. On each step of the service that executed the Petri nets, a new flag specific to each service was distributed to the corresponding service. Each flag contained (cryptographically) the service that it belonged to, the state of the service (active or not), and a timestamp signifying when the flag was created. Thus, when a flag was submitted by a team, the flag submission service had all the necessary information to determine the flag's validity (flags were valid for 5 minutes).

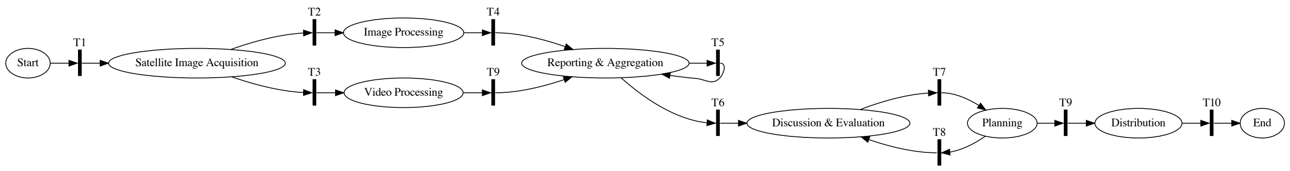
3.3.8 *Vulnerable Services*

There were 10 services in the iCTF, each service could be exploited only once per Petri net execution round; exploiting a service when it was not active resulted in an equal amount of negative points. Thus, to win the competition it was essential to understand and follow the missions. Table 1

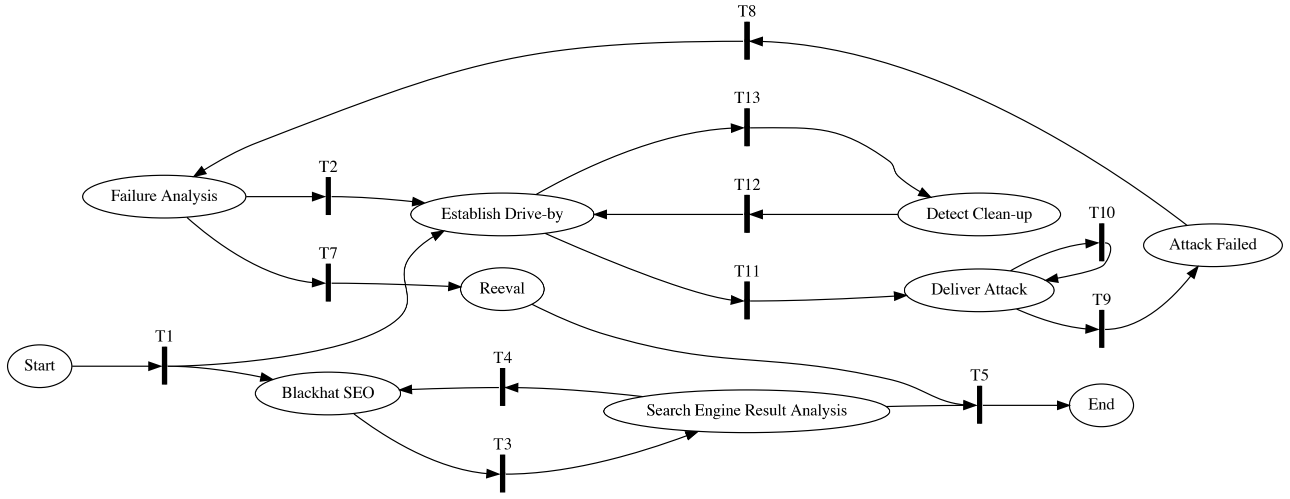
[1]`http://ictf.cs.ucsb.edu/data/ictf2010/final.gif`



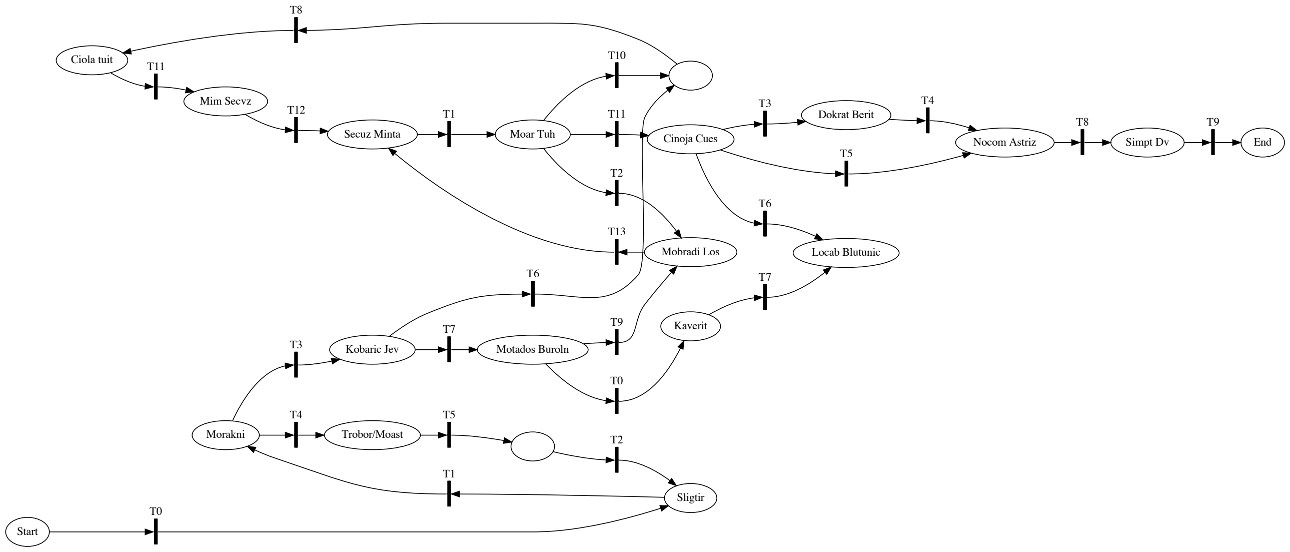
(a) CARGODSTR



(b) COMSAT



(c) DRIVEBY



(d) SEDAFER

Figure 2: Graphical representation of the missions given to the teams. The teams were actually given formats similar to Figure 1. Not shown here are the associations of the services to states in the Perti nets.

briefly summarizes the services. We direct the interested reader to Appendix A for an extended description of the services.

4. DATA ANALYSIS

In addition to being an excellent learning exercise for the teams involved, a security competition, if properly designed, can be a great source of data that is difficult to obtain in other contexts. In the iCTF competition, we created a game scenario to generate a Cyber Situational Awareness dataset.

Traffic collected during a security competition can be easier to analyze than real-world traffic, because there is more information about the network and participants in the competition. For example, all teams are identified, the vulnerable services are known, and there is no "noise traffic." Of course, a dataset collected in such controlled conditions also suffers from a lack of realism and is limited in scope. Nonetheless, the data collected during this competition is the first publicly available dataset that allows researchers to correlate attacks with the missions being carried out.

The iCTF competition generated 37 gigabytes of network traffic and complete information about services broken, challenges solved, flags submitted, bribes paid, IDS alerts, and bot connections. This data is made freely available[2].

As this is the first Cyber Situational Awareness dataset, many possibilities exist for its use in Situational Awareness research. One example would be using the dataset to train a host-based CSA intrusion detection system that could use more restrictive rules for a rule-based system (or tighter thresholds in an anomaly-based system) when a service is critical to a mission. One can also think of extending a host-based IDS to a network CSA intrusion detection system that understands not only the criticality of the services, but also their dependencies and relationships. Another example is the visualization of a network's activity with CSA in mind that helps a system administrator know which services are currently critical and which will become critical soon, helping them defend their network.

The firewall, bribing, bot, and money/points system can be viewed in a game theory light. The teams had to decide on the best way to allocate a scarce resource (money) to access the network and potentially win the game. The teams could perform any combination of bot connection and/or bribing to access the network. Further research could investigate how the choice of resource allocation affected each team's final result.

4.1 Description of Results

One problem with designing and implementing a novel competition is that teams may not understand the rules. This was a concern during the design of the iCTF competition. We worried that the novel aspects of the competition, especially the Petri net mission model, would be too complex for the teams to understand. However, when the first flags were submitted at 13:29, and subsequently when teams started submitting flags only for active services, it became apparent that many teams understood the competition.

Of the 72 teams, 39 submitted a flag, with 872 flags submitted in total. 48% may seem like a low number, however this means that almost half the teams broke at least one service. Many of the 39 teams submitted multiple flags, indicating that they understood the Petri net mission model.

Service	Total	Active	Inact.	% Inact.	Teams	Flags/Team
MostWanted	680	562	118	17	38	17.895
OvertCovert	97	82	15	15	6	16.167
IdreamOf.	49	37	12	24	6	8.167
WeirdTCP	24	23	1	4	2	12
LityaBook	16	12	4	25	3	5.333
icbmd	5	3	2	40	1	5
StolenCC	1	0	1	100	1	1

Table 2: Flags submitted per service.

At 17:00, "Plaid Parliament of Pwning" (PPP) of Carnegie Mellon University, took first place with 24,000 points. PPP submitted a total of 93 flags, with only 3 inactive flags (thus generating negative points), by compromising IdreamOfJeannie, MostWanted, and OvertCovert. Because PPP was able to compromise three services as well as understand the Petri net model (as evidenced by the submission of only three negative flags), they won first place.

Overall the teams exploited 7 of the 10 services: icbmd, IdreamOfJeannie, LityaBook, MostWanted, OvertCovert, StolenCC, and WeirdTCP. We believe this is because we underestimated the difficulty of the other 3 services. SecureJava and StormLog required a complex, multi-step process that proved too difficult for the teams to exploit. The teams also had trouble understanding the steps involved to exploit the session fixation vulnerability in LityaHot.

Table 2 describes the number of flags submitted for each service. The "Total" column is the total number of flags submitted for the service, "Active" and "Inact." are the number of flags that were submitted when a service was active or inactive. "% Inact." is the percent of flag submissions when the service was inactive. "Teams" shows the number of teams that submitted flags for the service and "Flags/Team" shows the average number of flags submitted per team.

MostWanted was the most exploited service, with 680 total flags submitted, followed by OvertCovert, with 97 flags submitted. It is clear that we did not estimate the difficulty of the services correctly, and, as evidenced by the number of teams that broke it, MostWanted was the easiest. Because the teams did not know the difficulty of the services, some luck is involved when teams decide which service to analyze first.

When we decided to create a complex competition, we knew that not every team would have the skills, experience, and luck to exploit a service and understand the Petri net mission model. However, we included 33 challenges in the competition of varying levels of difficulty and needing various skills to solve. We knew from past experience that even if a team couldn't exploit a service or understand the Petri net model of the missions, they would at least learn from (and enjoy) solving challenges. In fact, 69 out of 72 teams solved at least one challenge. Thus, even if a team was unable to exploit a service, they solved a challenge and hopefully had fun or learned something while competing in the iCTF.

4.2 Network Analysis

A benefit of designing a security competition is the ability to create an environment that allows for the testing of models and theories. By focusing the iCTF on Cyber Situational Awareness, we were able to create and evaluate Situational Awareness metrics. These metrics are applicable to many as-

[2]`http://ictf.cs.ucsb.edu/data/ictf2010/`

pects of CSA. We introduce *toxicity* and *effectiveness*, which are explained in the rest of this section.

First, we define three functions: $C(s,t)$, $A(a,s,t)$, and $D(s,t)$, each with a range of [0, 1]. Every function is specific to a service, s, and $A(a,s,t)$ represents an attacker, a.

$C(s,t)$ represents how critical a service, s, is with respect to time for a specific mission or set of missions. A value of 1 means that the service is very critical, while 0 means that the service is not critical.

$A(a,s,t)$ represents an attacker's, a, activity with respect to a service, s, throughout time. The value of the function is the perceived risk to the mission associated with the service. In most cases, the function has a value of 1 when an attack occurs and a value of 0 when there is no malicious activity. However, other, more complex models could be used (e.g., the type of attack could be taken into account).

$D(s,t)$ represents the damage to any *attacker* for attempting an attack on a service, s, at a given time, t. This function models the fact that every time an attack is carried out, there is a risk to the attacker, e.g., an intrusion detection system might discover the attack, the person using the targeted machine/service might notice unusual activity, etc.

We wish to define a metric, called *toxicity*, that captures how much damage an attacker has caused to a service over a time frame. Intuitively, it is the total amount of havoc the attacker has caused to the mission (or missions) associated with a service. *Toxicity* is calculated by first subtracting the damage to an attacker, $D(s,t)$, from the criticality of the service, $C(s,t)$. The resulting function, with a range of [-1, 1], describes at each point in time how much *any* attacker can profit by attacking at that moment. A negative value indicates that the attacker should not attack at that time.

The previously calculated function is general and has no bearing on a particular attacker. To calculate the damage caused by a specific attacker over time, we take the previously calculated function, $C(s,t) - D(s,t)$, and multiply it by $A(a,s,t)$. The resulting function, with a range of [-1, 1], shows how much damage a specific attacker caused to a given service. To calculate toxicity from this function, for a given time interval, t_1 to t_2, we take the integral of $A(a,s,t) * (C(s,t) - D(s,t))$ with respect to time. Equation (1) shows the calculation of the toxicity metric.

Toxicity is a measure for how much damage an attacker has caused to a given service, and can compare two attackers against the same service to see who did the most damage, however, it is specific to one service, and thus is useless as a comparison between a single attacker attacking multiple services or two attackers attacking different services. We propose *effectiveness* as a measure of how close an attacker is to causing the maximum toxicity possible. Intuitively, it is the ratio of the toxicity caused by an attacker to the toxicity an optimal attacker would cause. We define an optimal attacker as an attacker who attacks whenever $C(s,t)$ - $D(s,t)$ is positive, and this is shown in Equation (2). By substituting the optimal attacker in Equation (1) for $A(a,s,t)$, we obtain the formula for maximum toxicity, given in Equation (3). Taking the ratio of toxicity to maximum toxicity gives effectiveness, shown in Equation (4).

Toxicity, effectiveness, and $C(s,t)$, $A(a,s,t)$, and $D(s,t)$ can be used in future Cyber Situational Awareness research. By using the ideas presented here, an IDS could predict the behavior of an optimal attacker. Other tools could enable a network defender to perform "what-if" scenarios, seeing what would happen by increasing the damage to an attacker (e.g., by getting a new IDS), versus decreasing the criticality of the service (e.g., by getting a new server to perform the same function).

$$\mathit{Toxicity}(a,s,t_1,t_2) = \int_{t_1}^{t_2} A(a,s,t) * (C(s,t) - D(s,t))\,\mathrm{d}t \tag{1}$$

$$\mathit{OptimalAttacker}(s,t) = \begin{cases} 1 & \text{if } C(s,t) - D(s,t) > 0 \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

$$\mathit{MaxToxicity}(s,t_1,t_2) = \int_{t_1}^{t_2} \mathit{OptimalAttacker}(s,t) * (C(s,t) - D(s,t))\,\mathrm{d}t \tag{3}$$

$$\mathit{Effectiveness}(a,s,t_1,t_2) = \frac{\mathit{Toxicity}(a,s,t_1,t_2)}{\mathit{MaxToxicity}(s,t_1,t_2)} \tag{4}$$

The definitions of *toxicity* and *effectiveness* are general and apply to any arbitrary functions $C(s,t)$, $A(a,s,t)$, and $D(s,t)$. However, we constructed the iCTF competition so that we could measure and observe these functions and ensure they are valid metrics. We expected the higher ranked teams to show high toxicity and effectiveness for the services they broke.

The criticality, $C(s,t)$, of each service was defined in the following way: the function takes the value 1 when the service is active, and 0 when the service is inactive. Figure 3 shows the criticality graph for the most exploited service: MostWanted. When the function has a value of 1, one of the missions is in a state associated with the MostWanted service, otherwise the function has a value of 0. Note that for these and all the rest of the graphs of the competition, the X-axis is time, and starts at 13:30 PST, when the first flag was submitted, and ends at 17:00 PST, which was the end of the competition.

In our analysis, we define the damage to the attacker, $D(s,t)$, as the complement of the criticality graph, because if an attacker attacked a service when it was not active, they would get an equal amount of negative points. The damage graph alternates between 0 and 1, becoming 1 when the criticality is 0 and 0 when the criticality is 1. In our analysis, the criticality and damage functions are related as a byproduct of our design; however our definitions of toxicity and effectiveness do not depend on this; criticality and damage can be arbitrary and independent functions.

In order to calculate the toxicity of Plaid Parliament of Pwning against the various services, we must first calculate $A(a,s,t) * (C(s,t) - D(s,t))$ (note that this function has a range of [-1, 1]. Negative values in this context denote flags submitted when a service was inactive). This is shown in Figure 4 for the service MostWanted, and Figure 5 for the service OvertCovert. As can be seen in Figure 4, PPP did not attack at the incorrect time for the MostWanted service, but submitted several incorrect flags for OvertCovert, as evidenced by the negative values in Figure 5.

Toxicity is calculated by taking the integral of this function between 13:30 and 17:00 PST. However, since the time in-between each flag change is a random value between 60 and 120 seconds, and a team is able to exploit the service only once per flag change, we simplified the time between

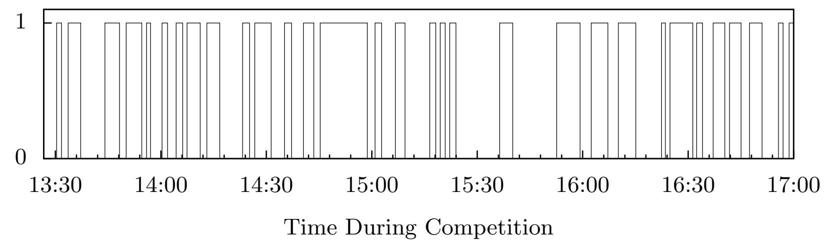


Figure 3: $C(s, t)$ of the service MostWanted.

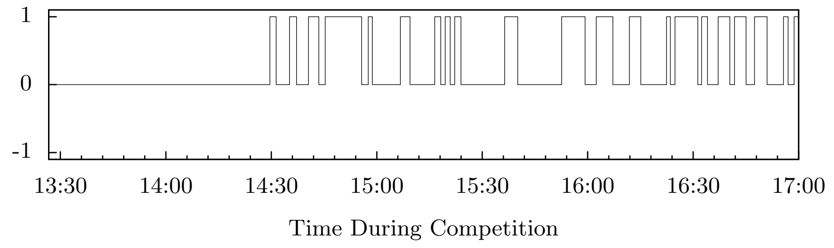


Figure 4: $A(a, s, t) * (C(s, t) - D(s, t))$ of team PPP against the service MostWanted.

flags as 1, which returned a round number for the toxicity metric. In the general case, however, the amount of time a service is critical is very important for calculating toxicity and should not be oversimplified. Because the criticality of our services changed at discrete intervals, we are able to make this simplification without adversely affecting our results.

Table 3 shows the toxicity and effectiveness of the top 5 teams for each of the services that were successfully exploited. The results are as we expected; many of the most effective teams placed high in the final rankings. The first place team, PPP, team #113, was not only the most effective for three different services: IdreamOfJeannie, MostWanted, and OvertCovert, but, also, with 65% effectiveness on MostWanted, had the highest effectiveness of any team. PPP's dominance is apparent because they did not just break three services, but they were also highly effective. The second place team, 0ld Eur0pe (team #129), was the second most effective at IdreamOfJeannie and third most effective at MostWanted.

5. LESSONS LEARNED

For this edition of the iCTF competition, we tried to capitalize on our previous experience by learning from mistakes of years past. However, we may hope to the contrary, we are still human: we made some mistakes and learned new lessons. We present them here so that future similar competitions can take advantage of what worked and avoid repeating the same mistakes.

5.1 What Worked

The pre-competition setup worked extremely well. Having the teams connect to the VPN and host their own VMware bot image was helpful in reducing the support burden on the day of the competition, where the time is extremely limited.

In the past, having a complex competition frustrated many teams and caused them to spend a substantial amount of time trying to figure out the competition instead of actually competing. To combat this, we released details about the structure of the game, the Petri net models of the missions, and the Snort configuration in advance. We hoped that this would give teams the opportunity to come to the competition well-prepared. Another advantage in giving advance notice is that it rewards teams who put in extra time outside of the eight hours of the competition. This is important, as the larger part of the education process is actually associated with the preparation phase, when students need to become familiar with different technologies and brainstorm possible attack/defense scenarios.

Another positive feedback we received through informal communication was that the theme of the competition was clear and consistent. The iCTF competition has always had a well-defined background story, which supports understanding and provides hints on how to solve specific challenges. People explicitly appreciated the effort put into creating a consistent competition environment and complained about competitions that are simply a bundle of vulnerable services to exploit.

From the comments of the players, it was clear that a substantial amount of effort was put into preparing and developing the right tools for the competition. This is one of the most positive side-effects of the participation in this kind of live exercises. Having to deal with unknown, unforeseen threats forces the teams to come up with general, configurable security tools that can be easily repurposed once the focus of the competition is disclosed. The continuous change in the iCTF design prevents the "overfitting" of such tools to specific competition schemes.

In general, through the past three years we found that radical changes in the competition's design helped leveling the playing field. Although the winning teams in the 2008,

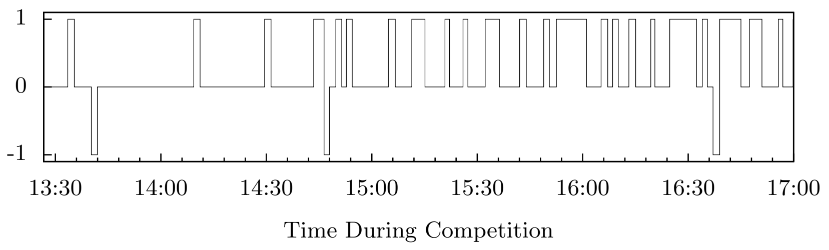


Figure 5: $A(a,s,t) * (C(s,t) - D(s,t))$ of team PPP against the service OvertCovert.

Service	Team	Toxicity	Effectiveness	Service	Team	Toxicity	Effectiveness
icbmd	126	3	0.03896	MostWanted	113	42	0.65625
icbmd	124	1	0.01298	MostWanted	114	40	0.625
IdreamOfJeannie	113	14	0.23728	MostWanted	129	36	0.5625
IdreamOfJeannie	129	12	0.20338	MostWanted	105	34	0.53125
IdreamOfJeannie	123	10	0.16949	MostWanted	152	30	0.46875
IdreamOfJeannie	111	2	0.03389	OvertCovert	113	36	0.48648
IdreamOfJeannie	128	-6	-0.10169	OvertCovert	131	16	0.21621
LityaBook	149	8	0.11428	OvertCovert	123	10	0.13513
LityaBook	166	5	0.07142	OvertCovert	117	9	0.12162
LityaBook	150	-1	-0.01428	OvertCovert	127	2	0.02702
LityaBook	137	-2	-0.02857	WeirdTCP	156	13	0.23214
StolenCC	123	1	0.02040	WeirdTCP	105	6	0.10714
StolenCC	105	1	0.02040				
StolenCC	152	0	0.0				

Table 3: Top 5 most effective teams per service.

2009, and 2010 editions were still experienced groups, teams of first-time competitors placed quite high in the ranking. This was possible because we intentionally did not disclose in advance to the teams the nature of these new competitions. Many "veteran" teams expected a standard CTF and were surprised to learn that this was not the case. Of course, it is hard to keep surprising teams, as designing new competitions requires a substantial amount of work. However, it is arguable that this type of competition is inherently easier for novice teams to participate in.

Finally, the competition generated a unique, useful dataset that can be used to evaluate cyber situation awareness approaches. This aspect of security competitions cannot be overemphasized, as a well-designed data-capturing framework can provide a wealth of useful data to security researchers.

5.2 What Did Not Work

LityaLeaks, the part of the infrastructure used to distribute the fired transitions of the Petri nets, as well as various hints and clues about services and challenges, was an integral part of our design (and the name fit in nicely with the theme). However, using a base MediaWiki [9] installation on a virtual machine with 256 MB of RAM was a mistake. As soon as the competition started, LityaLeaks was brought to a crawl due to the amount of traffic created by the teams.

Having LityaLeaks down was very problematic, because if teams couldn't see which transitions were firing then they couldn't submit flags. Eventually, a static mirror of LityaLeaks was brought up. Because of this, we had to change the Petri net software on the fly to update a publicly accessible file with the transition firings instead of using LityaLeaks. Once the change was made, at 13:30 PST, teams started submitting flags, and the rest of the competition went fairly smoothly.

As the scoreboard is the only way for teams to understand the current state of the game, making the scoreboard accurately reflect the status of the competition was essential. However, each piece of the competition's infrastructure was developed and tested independently. Knowing that getting the firewall, mothership, and Snort systems working properly was very important, those parts of the functionality were heavily tested in isolation. However, the interaction of these systems with the scoreboard was not tested before the competition. Thus, during the competition we discovered that the reasons given to teams for being blocked on the scoreboard were not correct, and in some instances the connection status of some teams were incorrect. Due to one of the developers being ill, it took us most of the competition to completely resolve this issue. While we were fixing the issue, we communicated to teams that to test their network connectivity, they could simply try connecting to one of the services. In the future, we will be testing our infrastructure as a whole, including important pieces like the scoreboard.

One issue with creating a complex and novel competition is that some teams might not "get" the competition. This can be on a number of levels, perhaps the team has never heard of Petri nets or could not exploit any of the services. This puts them at an extreme disadvantage in the rankings, as they cannot score any points. This was the case for 33 teams. However, for the 39 teams that submitted flags, a novel competition challenged them to create new solutions and tools, learning in the process. Ultimately, it is up to the competition administrators to balance novelty, complexity, and fairness.

5.3 What Worked?

Putting a backdoor into the bot VM that we distributed to the teams was something that we implemented five hours before the distribution of the VM. Something that we saw as funny turned out to have serious implications. One team came to us and said that they had an exploit to reduce every team's money to zero, effectively removing everyone else from the competition. Using the backdoor, they could bribe the Litya administrators as the team's bot, thus draining all of the team's money. We asked them not to do this, as it was unsporting to completely shut off most team's access to the services, and fixed this avenue of attack. We also alerted the teams to the existence of a backdoor on their VMs. Later in the competition, a team came to us complaining that their points kept decreasing. Looking into it, a team was exploiting a service, and submitting all the inactive flags (worth negative points) through another team's compromised bot. The team that this happened to came in last place (with -3300 points).

The backdoor provided some interesting (and funny) situations, however it came at a price. The last place team felt that this was an unsporting thing to do and were rightly upset over their last-place standing. We ruled that, since we had given notice about the backdoor, and given the extremely easy fix (filter the traffic from other teams), the outcome was acceptable. However, this situation did highlight an issue that these kind of "easter eggs" can produce: while it may be exciting and interesting for the teams who discover them, the more inexperienced teams who are not looking for them and/or can't find them are put at a disadvantage. This just increases the gap between the experienced and inexperienced.

6. CONCLUSIONS

Live cyber-security exercises are a powerful educational tool. The main drawback of these exercises is that they require substantial resources to be designed, implemented, and executed. It is therefore desirable that these exercises provide long-lasting byproducts for others to use for security research. In this paper, we presented a unique, novel design for a live educational cyber-security exercise. This design was implemented and a competition involving almost a thousand world-wide students was carried out in December 2010. We discussed the lessons learned, and we presented the dataset we collected, which we believe is the first public dataset focused on Cyber Situational Awareness. We hope that this dataset will be useful to other researchers in this increasingly popular field and that future security exercises will yield interesting datasets.

7. REFERENCES

[1] T. Augustine and R. Dodge. Cyber Defense Exercise: Meeting Learning Objectives thru Competition. In *Proceedings of the Colloquium for Information Systems Security Education (CISSE)*, 2006.

[2] N. Childers, B. Boe, L. Cavallaro, L. Cavedon, M. Cova, M. Egele, and G. Vigna. Organizing Large Scale Hacking Competitions. In *Proceedings of the Conference on Detection of Intrusions and Malware and Vulnerability Assessment (DIMVA)*, Bonn, Germany, July 2010.

[3] W. Clark. *The Gantt chart: A working tool of management.* New York: Ronald Press, 1922.

[4] C. Cowan, S. Arnold, S. Beattie, C. Wright, and J. Viega. Defcon Capture the Flag: defending vulnerable code from intense attack. In *Proceedings of the DARPA Information Survivability Conference and Exposition*, April 2003.

[5] A. D'Amico, L. Buchanan, J. Goodall, and P. Walczak. Mission Impact of Cyber Events: Scenarios and Ontology to Express the Relationships between Cyber Assets, Missions and Users. In *Proceedings of the International Conference on Information Warfare and Security*, Dayton, Ohio, April 2010.

[6] D. R. Hipp. Sqlite. `http://www.sqlite.org/`, 2010.

[7] Justin.tv. `http://justin.tv/`.

[8] S. Liang. *Java Native Interface: Programmer's Guide and Reference.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1st edition, 1999.

[9] Mediawiki. `http://www.mediawiki.org/`.

[10] B. Mullins, T. Lacey, R.Mills, J. Trechter, and S. Bass. How the Cyber Defense Exercise Shaped an Information-Assurance Curriculum. *IEEE Security & Privacy*, 5(5), 2007.

[11] J. Peterson. Petri Nets. *ACM Computing Surveys*, 9(3), September 1977.

[12] L. Pimenidis. Cipher: capture the flag. `http://www.cipher-ctf.org/`, 2008.

[13] Pwn2own 2009 at cansecwest. `http://dvlabs.tippingpoint.com/blog/2009/02/25/pwn2own-2009`, March 2009.

[14] W. Schepens, D. Ragsdale, and J. Surdu. The Cyber Defense Exercise: An Evaluation of the Effectiveness of Information Assurance Education. Black Hat Federal, 2003.

[15] Simpp. 3vilsh3ll.c. `http://packetstormsecurity.org/files/view/64687/3vilSh3ll.c`.

[16] Snort. `http://www.snort.org/`.

[17] B. Stone-Gross, R. Abman, R. Kemmerer, C. Kruegel, D. Steigerwald, and G. Vigna. The Underground Economy of Fake Antivirus Software.

[18] The HackerDom Group. The ructf challenge. `http://www.ructf.org`, 2009.

[19] G. Vigna. Teaching Hands-On Network Security: Testbeds and Live Exercises. *Journal of Information Warfare*, 3(2):8–25, 2003.

[20] G. Vigna. Teaching Network Security Through Live Exercises. In C. Irvine and H. Armstrong, editors, *Proceedings of the Third Annual World Conference on Information Security Education (WISE 3)*, pages 3–18, Monterey, CA, June 2003. Kluwer Academic Publishers.

[21] VMware. `http://www.vmware.com/`.

APPENDIX

A. VULNERABLE SERVICES

A brief description of the 10 services in the iCTF and the vulnerabilities associated with it follows.

LityaBook was a social networking website, similar to Facebook. By creating an underage girl profile, the attacker would cause President Bironulesk to visit their profile. They could then use a Cross-Site Scripting attack to steal President Bironulesk's browser's cookie, which contained the flag.

LityaBook also had a session fixation vulnerability. The authentication cookie contained the MD5 of the session ID. Therefore, an attacker could lure a victim to log in with a specific session ID, allowing an attacker to impersonate the victim. This vulnerability could have been exploited by using another website, LityaHot.

LityaHot was a website where young models posted links to their pictures, waiting for casting agents to contact them. Periodically, a member of President Bironulesk's staff, Femily Edeo, visited this site, clicking on links people had posted. If the link was a LityaBook page, he logged in to check the pictures. Thus an attacker could post a link on LityaHot, leveraging the session fixation vulnerability to log into LityaBook as Edeo and obtain the flag.

icbmd was the first iCTF service with perceptible effects on the real world. A USB foam rocket launcher was connected to a control program, pointing in the direction of a physical target. A time-sharing mechanism was used to share the missile launcher amongst the teams. Each team had a visual clue of where the launcher was aiming, via a webcam mounted on the missile launcher with a live streamed video to the Justin.tv on-line video streaming service [7]. The team currently controlling the missile launcher could exclusively connect to the control and move the launcher's turret. An encoded version of the launch code was leaked to the teams. After deciphering the code, the teams were able to launch a missile. Once a team successfully hit the target, the flag was sent to them.

StormLog was a web application that displayed log files generated by a fake botnet called "Storm." This service had a directory traversal vulnerability which allowed an attacker to download a copy of the cgi-bin program. An attacker had to exploit an off-by-one overflow in the cgi-bin program to execute arbitrary code and obtain the flag.

StolenCC was a web service that displayed text files containing credit card numbers. The cgi-bin program was written in Perl and contained a directory traversal vulnerability. By inserting a null character into the `filename` parameter, an attacker could bypass the program's sanity checking and open any file. Then, an attacker could use additional functionality of Perl's `open` to execute any command, finding and displaying the flag.

SecureJava was a web service that used a Java applet to perform authentication. An attacker needed to get past the authentication to find the flag. This involved reverse engineering the encryption algorithm. Once understood, the attacker leveraged a flaw in the encryption algorithm to steal the flag.

IdreamOfJeannie was a Java service that collected credit card information. Even though the bulk of the service was written in Java, JNI [8] was used to include a function written in C, which contained an off-by-one error. The attacker could utilize the off-by-one error to obtain the flag.

WeirdTCP was a C service that acted as a file server with a trust relationship with a specific IP address. A blind TCP spoofing attack against the service pretending to be the trusted IP address was required to find the key. However, due to the VPN technology we were using, packets could not be spoofed. A custom IP protocol RFC was given to the teams, which introduced an IP option that could be used to overwrite the source address of an IP packet. Thus an attacker had to use the IP option to spoof the trusted IP address, and, in addition, perform a sequence number guessing attack, in order to provide the correct acknowledgment number during the TCP handshake. Once the TCP connection was established, the attacker received the flag.

MostWanted was a Python service with a SQLite [6] backend. The service hosted mugshots of various wanted "criminals," and allowed a user to create or view mugshots. MostWanted had a stored SQL-injection vulnerability, which an attacker had to exploit to access the flag.

OvertCovert was a C-based service that allowed a user to store and access encrypted data. An attacker had to first exploit a printf vulnerability (which disallowed `%n`) to extract the encryption key. Then, an off-by-one error was used to access the encrypted flag. Using the key previously obtained, the attacker could decrypt the flag and exploit the service.

"Mix-In-Place" Anonymous Networking Using Secure Function Evaluation

Nilesh Nipane, Italo Dacosta and Patrick Traynor
Converging Infrastructure Security (CISEC) Laboratory
Georgia Tech Information Security Center (GTISC)
Georgia Institute of Technology
{nnipane3, idacosta, traynor}@cc.gatech.edu

ABSTRACT

Anonymous communications systems generally trade off performance for strong cryptographic guarantees of privacy. However, a number of applications with moderate performance requirements (e.g., chat) may require both properties. In this paper, we develop a new architecture that provides provably unlinkable and efficient communications using a single intermediary node. Nodes participating in these Mix-In-Place Networks (MIPNets) exchange messages through a mailbox in an Oblivious Proxy (OP). Clients leverage Secure Function Evaluation (SFE) to send and receive their messages from the OP while blindly but reversibly modifying the appearance of all other messages (i.e., mixing in place) in the mailbox. While an Oblivious Proxy will know that a client participated in exchanges, it can not be certain which, if any, messages that client transmitted or received. We implement and measure our proposed design using a modified version of Fairplay and note reductions in execution times of greater than 98% over the naïve application of garbled circuits. We then develop a chat application on top of the MIPNet architecture and demonstrate its practical use for as many as 100 concurrent users. Our results demonstrate the potential to use SFE-enabled "mixing" in a single proxy as a means of providing provable deniability for applications with near real-time performance requirements.

1. INTRODUCTION

The lack of privacy on the Internet is well documented. Whether by neighbors [55], ISPs [60], advertisers [65] or governments [54], most online communications are easily observable by parties other than the source and intended destination. Unfortunately, the use of encryption is often not enough to guarantee privacy or anonymity. Simply knowing that two endpoints exchange messages may provide sufficient context to reveal sensitive information. Specifically, traffic analysis has repeatedly been demonstrated as an effective means of uncovering not only individual relationships, but also larger structures in corporate [44], social [35] and other [13] organizations.

Private communication over open networks has been studied for more than 30 years. A range of systems have been created during this time across a design space that includes tradeoffs in privacy guarantees, performance and threat model and has resulted in two general anonymous communication architectures – relay-based and superposed. The best known of the former is Tor [17], which is the intellectual descendent of Chaum's mix architecture [10]. Tor delivers iteratively encrypted messages through a series of proxies and achieves high levels of privacy in the presence of a large amount of cross traffic. This approach also provides high performance, making it ideal for applications such as web surfing. Superposed communications architectures provide cryptographically-strong guarantees of privacy without the multi-hop and cross traffic assumptions of systems such as Tor, but do so at great expense to performance. Accordingly, such systems are generally more appropriate for very low bandwidth applications such as ballot casting. An architecture somewhere between these two classes, in which the strong guarantees of superposed systems and the performance of relay-based systems without the need for multiple intermediaries could help facilitate anonymous communications for different classes of applications (e.g., private messages between members of a corporate board, deniable chatting between an executive and a recruiter in a coffee shop, etc). This architecture addresses that void.

In this paper, we develop a protocol and supporting infrastructure capable of providing strong guarantees of sender and receiver anonymity for applications with moderate performance constraints (e.g., chat). *Mix-in-Place Anonymous Networks* (MIPNets) attempt to retain the intuitive security properties of traditional mix networks while reducing the attack surface introduced by requiring the use of multiple intermediary proxies. Using *Secure Function Evaluation* (SFE), clients are able to send and are restricted to receive a subset of the messages stored in an *Oblivious Proxy* (OP). These exchanges also blindly but reversibly modify the appearance (but not the contents) of all other messages stored in the OP, making it appear to an adversary that the contents of all messages are overwritten during each exchange. Accordingly, an adversary is unable to determine which messages are actually modified, yet alone their source and destination. Whereas traditional mix networks rely on a cascade of nodes, *this approach achieves unlinkability by mixing communications in place through a cascade of functions, constantly and indefinitely perturbing the appearance of messages in the system.*

Intuitively, MIPNets work as follows. Clients use SFE to read and write to a central mailbox (the OP) in a round-robin fashion. Because each exchange changes the appearance of all messages in the mailbox, individual interactions can not be determined by the OP or any other party. However, realizing this seemingly simple system presents a number of significant challenges. First, our proposed approach must function like a superposed architecture and prevent the kinds of side-channel attacks that have made identification possible in current mix networks. We achieve this by building our architecture based on a ring topology, thereby excluding the majority of timing, temporal perturbation and frequency-based attacks. We also take a number of steps to make this architecture robust against actively malicious participants, who may attempt to pollute the contents of messages or deny service to other clients within the MIPNet. Finally, we significantly improve the performance of the underlying cryptographic operations and implement a number of optimizations in the hopes of making this architecture usable by classes of applications requiring strong guarantees of anonymity in environments where sufficient cross traffic is not a given and better performance than superposed systems.

We address these challenges through the following contributions:

- **Develop an architecture that provably guarantees the unlinkability of communications between participants:** We present a new architecture for anonymous communication based on Secure Function Evaluation. While a number of provably anonymous communications systems have been proposed previously, *this is the first use of SFE in this domain.* Nodes send and receive messages through an Oblivious Proxy, which is unaware of their contents, destination or veracity. Moreover, the appearance of all messages is blindly altered after each exchange, further obfuscating the attacker's view of communications.

- **Improve and tune performance of SFE primitives:** We build and characterize our architecture using a modified version of Fairplay [42]. We not only implement a more efficient oblivious transfer primitive for Fairplay, but also explore numerous compilation and run-time options to reduce execution time over the naïve use of SFE in our architecture by as much as 98.5%, *allowing these heavyweight cryptographic operations to form the basis of practical, performance conscious, privacy-preserving applications.* The use of these cryptographic primitives as the basis of near real-time applications has not previously been considered possible.

- **Implement and measure core architecture and build an anonymous instant messaging application:** We perform an extensive performance analysis of our proposed architecture. We then construct a sample instant messaging client running on top of our system and demonstrate the ability to process keypresses *as fast as they are entered* for groups of as many as 100 users.

We note that MIPNets are not intended to be a replacement for traditional mix networks such as Tor [17]; rather, they target a different portion of the application space. MIPNets are designed to provide higher assurance of anonymity for classes of applications that require it (e.g., chat in hostile networks with low cross traffic) and are willing to make tradeoffs to achieve these guarantees.

2. RELATED WORK

Anonymity in computer networks has been studied since the inception of such networks. Outside of systems with a trusted and centralized authority [2], two classes of solutions have arisen. Note that these solutions differ significantly from trusted anonymous proxies [8, 56, 78], which operate as a single-hop filtering point and can therefore potentially be coerced into revealing the link between a source and destination, and from anonymous publication and storage systems such as Freenet [12], Free Haven [16] and Publius [73].

Applying cryptographic modifications to traffic through a proxy or *mix* in an untrusted network was first proposed by Chaum [10]. In this scheme, clients select a series of mix nodes, called a *cascade*, through which their messages should pass. Each message is then encrypted with the public key of each mix in the reverse order of the path to its destination. As such messages traverse the cascade, each mix node decrypts the message and exposes the next layer of the encrypted packet and forwarding instructions. While originally suggested for store and forward protocols, the mix architecture was eventually realized for all traffic via Onion Routing [59, 68]. This approach also motivated the creation of a number of other related efforts [51, 30, 14, 25, 6, 24, 15, 47, 20], with Tor [17] being the most widely used and studied of all such systems. Extensions to these schemes include the use of universal reencryption [27, 29], steganography [31] and mix rings for performance [9]. Crowds [61] and Hordes [39] similarly forward packets among a series of nodes.

While providing anonymity against a number of adversaries [69, 21], mix networks have become the target of an increasing number of attacks. These attacks introduced a number of techniques allowing adversaries with limited knowledge and control of a network to link sender and receiver. Through attacks on timing [79, 76, 38, 49, 74, 48, 64, 57, 75, 67] and other vectors [43, 3], such identification may be practically possible.

Chaum also suggested a second class of anonymous communications mechanisms with the Dining Cryptographers Problem [11]. This technique allows a sender to anonymously transmit a single bit as follows: Alice and Bob (who wants to transmit a single bit to Charles) flip a coin in secret. Alice reports the result of the flip to Charles. As the message Charles receives will be the XOR of all of the bits reported by the participants, Bob's report of the coin flip depends on the bit that he wishes to send. For instance, if Alice reports 0 (heads) and Bob wishes to transmit a 1 (tails), Bob claims the flip resulted in a 1. From Charles' perspective, each party is equally likely to have lied about the result, thereby protecting the sender's identity. A number of systems have attempted to implement such *DC-net* protocols through a variety of communications mechanisms [52, 18, 63] and with resistance to collisions and maliciousness [72, 7, 71, 70, 28]. Herbivore [66] addresses many of these issues by broadcasting all messages to all hosts across a star topology and scales through the use of cliques. Such networks are susceptible to denial of service and statistical analysis attacks, wherein nodes in different cliques communicating frequently can be correlated with higher accuracy. The Pynchon Gate [62] and pMixes [45] use Private Information Retrieval (PIR) protocols to achieve receiver anonymity. Senders place messages into mailboxes in a common node where receivers use PIR to retrieve them without a passive adversary being able to determine the content delivered to the destination. However, this approach is limited in a number of ways, including that sender anonymity is not guaranteed and the potential for timing correlation attacks.

3. SECURE FUNCTION EVALUATION

3.1 Background

Before presenting the details of the MIPNet architecture, we provide an overview of the cryptographic primitives used to communicate between clients and the OP.

Secure Function Evaluation [42, 37, 1, 26, 22, 23, 40, 53, 41, 33] allows two or more parties to execute a joint computation traditionally requiring the oversight of a trusted third party without any external intervention. More formally, participants Alice and Bob have input vectors $\vec{a} = a_0 \cdots a_{n-1}$ and $\vec{b} = b_0 \cdots b_{m-1}$ and wish to learn $f(\vec{a}, \vec{b})$ without revealing any information about their inputs that can not be inherently inferred from the output of $f(\vec{a}, \vec{b})$. As an example, SFE creates "circuits" that solve the Millionaires' problem [77], wherein two millionaires want to know who has more money without revealing their actual wealth. Assuming that each millionaire enters their net worth honestly to a function asking the high level question "Do I have greater worth than the other participant?", SFE can return the correct Boolean values to each party without disclosing either input.

While these cryptographic mechanisms have been understood for decades, their uptake and use in real systems has been slow. In particular, the use of such protocols has long required in-depth knowledge of the underlying cryptographic constructions and compelled interested parties to implement their own libraries. The Fairplay compiler [42, 5] greatly simplifies such efforts. Fairplay allows programmers to specify their protocol in a high-level language and then creates a garbled/encrypted Boolean circuit encoding a SFE-version of the desired function. In an exchange between Alice and Bob, Bob uses Fairplay to encode circuits for himself and then transmits a set to Alice. Both parties input their data, exchange their results and then uses them to compute the answer.

The security of these operations is guaranteed through the use of *Oblivious Transfers* (OT) [58, 77, 50]. A 1-out-of-n OT protocol allows Bob to learn one of the n pieces of data possessed by Alice without Alice learning the identity of the specific object. OT protocols also allow for Alice to prevent Bob from learning the contents of the remaining $n-1$ pieces of data. Fairplay implements an optimized 1-out-of-2 OT protocol proposed by Naor and Pinkas [50], which is based on the Diffie-Hellman problem. This protocol is secure in the random oracle model, which is implemented using the SHA-1 hash function.

Two parties engaging in an SFE exchange using Fairplay perform a single OT for each input into the circuit. After these exchanges, Alice is able to evaluate the garbled circuit without further interacting with Bob. Malicious behavior by Alice is defended against by both the security of the OT protocol and the use of SHA-1 to encode the secret used to encrypt the circuit. The same constructions also prevent Bob from malicious behavior, assuming that the circuits encoded by Bob are correct.

The authors of Fairplay recommend the use of a cut-and-choose protocol in the event that Bob's creation of the circuits can not be trusted. In particular, Bob can be required to create m garbled circuits for the function f and send them to Alice. Alice then randomly selects the circuit she wishes to use for her exchange and requests the secrets corresponding to the $m-1$ other circuits. Alice can then independently evaluate the correctness of the ungarbled circuits and can execute the remaining circuit with probability $\frac{1}{m}$ of maliciousness.

3.2 Extensions to Fairplay

Fairplay not only provides an OT primitive on top of which SFE can be built, but also makes it straightforward to implement supplemental OT schemes should improvements become available. We implemented an additional k-out-of-n scheme based on a Two-Lock cryptosystem. Such schemes are often more efficient than related OT mechanisms.

A Two-Lock cryptosystem is defined as follows: Assume that there are two encryption algorithms A and B. Alice encrypts a message m with secret s using scheme A and Bob encrypts the same message with key s' using scheme B. If $A_s(B_{s'}(m)) = B_{s'}(A_s(m))$, then Alice and Bob can be ensured that the contents of the messages they exchange remain confidential without actually sharing a key. Using this scheme, Alice encrypts a message m and sends the quantity $X = A_s(m)$ to Bob. Bob encrypts the received message, resulting in $Y = B_{s'}(X)$, which he then sends to Alice. Alice decrypts Y, creating $C = A_s^{-1}(Y)$ and again sends this value to Bob. Bob then decrypts C such that $m = B_{s'}^{-1}(C)$, thereby recovering the message m initially encrypted by Alice.

The above method can be used to perform k-out-of-n OT as follows. Alice encrypts n messages such that $X_0 = A_s(m_0), \cdots, X_{n-1} = A_s(m_{n-1})$. Bob selects a subset k of the n messages and encrypts them as $Y_0 = B_{s'}(X_{k_0}), \cdots, Y_{k-1} = B_{s'}(X_{k_{k-1}})$. Alice then receives the k quantities of Y, which she can not identify because Bob has encrypted them, and decrypts them resulting in $C_0 = A_s^{-1}(Y_0), \cdots, C_{k-1} = A_s^{-1}(Y_{k-1})$. Finally, Bob decrypts $C_0, \cdots, C_{k-1}$ and reveals the k selected messages.

Two-Lock cryptosystems can be implemented based on either the RSA or Discrete Log problems. While both approaches provide sufficient protection of the encrypted data, systems based on the RSA problem can be made to run more efficiently through the use of small values of e. Because the 0s and 1s are encoded as long strings of bits of length greater than $|N|$ in Fairplay, this optimization is not susceptible to small exponent attacks. From the RSA-based scheme proposed by Huang et al. [32], Alice sends Bob $X_0 = m_0^e, \cdots, X_{n-1} = m_{n-1}^e$. Bob selects k random numbers, one for each message he wishes to ultimately receive, and sends Alice $Y_0 = X_{k_0} \cdot k_0^e, \cdots, Y_{k-1} = X_{k_{k-1}} \cdot k_{k-1}^e$. Alice responds by sending Bob $C_0 = Y_0^d, \cdots, C_{k-1} = Y_{k-1}^d$, from which Bob recovers the messages $m_{k0} = k_0^{-1} \cdot C_0, \cdots, m_{kk-1} = k_{k-1}^{-1} \cdot C_{k-1}$.

The 1-out-of-2 OT scheme implemented with Fairplay requires a total of three modular exponentiations by the sender and two modular exponentiations by the receiver. Moreover, this approach requires two random number generations and a total of four message transfers. The RSA-based scheme also requires three modular exponentiations by the sender, but these operations are faster given our use of small values of e. The receiver side need only perform a single modular multiplication and one modular division. This approach also requires only a single random number generation and also requires the exchange of four messages. Accordingly, we expect to see a slight but measurable difference in the per-OT efficiency of our approach. We present our experimental results in Section 5.

4. SYSTEM ARCHITECTURE

In this section, we present the details of the MIPNet architecture. We begin by discussing the intuition behind our design and formalize the desired system guarantees. We then present a simple version of our architecture that protects against a globally passive adversary and assumes that all participants in the network are benign. Our second model provides additional protections against active attackers. We then discuss methods for graceful scaling and characterize the probability of collisions in the second system and prove that the previously defined guarantees hold in MIPNets.

Note that we focus on the design of the MIPNet architecture and, for the time being, assume that all clients participate in the network for long periods of time. Due to space limitations, we discuss membership management issues in greater detail in our technical report.

4.1 Design Intuition

Mix networks are an attractive model for anonymous communications for a number of reasons. The use of multiple layers of encryption offers an intuitive mechanism for obscuring the relationship between source and destination. However, it is often what happens *between proxies* in such systems that allows an adversary to link two endpoints. For instance, by injecting additional traffic into a proxy suspected to be in the path of a specific flow, an adversary can observe increased latency between source and des-

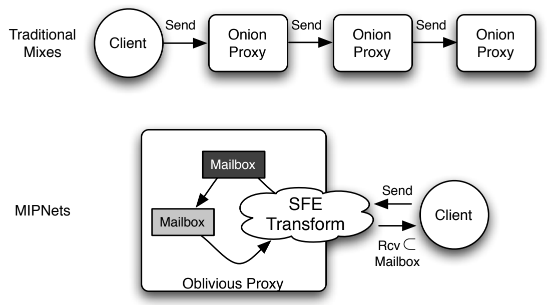


Figure 1: A conceptual comparison of traditional mix networks and MIPNets. Note that at every "Send", an adversary has an opportunity to perturb traffic. Mixing in place allows for such attack vectors to be largely removed.

tination [19]. Moreover, flows can be watermarked by perturbing packet interarrival times, allowing the adversary to identify the participants of a flow with a high degree of accuracy without injecting additional traffic. While the use of multiple intermediaries reduces the probability of a single proxy being able to determine the sender and receiver, it also increases the attack surface of the entire system.

The MIPNet architecture attempts to reconcile these issues by *eliminating the need to use multiple proxies*. The use of only a single proxy would appear flawed as no number of layers of encryption would prevent a proxy in possession of all of the necessary keys from linking source and destination. However, this issue can be overcome through the use of SFE between clients and the proxy. Instead of giving the OP messages that it can decrypt, interpret and forward, we simply require it to hold messages in a mailbox. Clients use SFE exchanges not only to send and retrieve the message stored within a subset of the slots in the mailbox, but also to decrypt and reencrypt the entire mailbox. Because of the properties of SFE, this final operation can take place without clients revealing the key to the OP or the OP revealing the entirety of the mailbox to each client. It is this cascade of *functions* that provides for anonymous communications in MIPNets. This "middle-path" allows us to enjoy much of the simplicity of relay-based anonymous communication while not having to rely on multiple proxies. Figure 1 highlights this difference.

The work on anonymous buses [4] and flash mixes [34] are the most similar related efforts to ours; however, our approach is novel in a number of ways. Unlike the former, the use of SFE makes it so that no client is ever able to see the entire contents of the mailbox. Second, while buses are secure in the honest-but-curious model, our solution is secure in the presence of a malicious adversary. Unlike both schemes, our approach also prevents the mixing node from knowing the relationship between messages before and after mixing. Finally, the performance evaluation and parameter characterization of our system is in much greater depth than previous works.

4.2 Desired System Properties

More formally, our system strives to provide the following guarantees: Let $\mathcal{C}$ be the universe of all possible clients and $\mathcal{S_A} \in \mathcal{C}$ be the set of clients currently connected to the OP.

Definition 1. Sender Anonymity: Let client $c \in \mathcal{S_A}$ be the sender of message m_i, denoted $c = sender(m_i)$. A system provides sender anonymity iff a passive adversary can determine that $c \in \mathcal{S_A}$ and $sender(m_i) \in \mathcal{S_A}$, but *not* that $c = sender(m_i)$ with probability greater than $\frac{1}{|\mathcal{S_A}|} + \epsilon$, where ϵ is a negligible value.

Definition 2. Receiver Anonymity: Let client $c \in \mathcal{S_A}$ be the receiver of message m_j, denoted $c = receiver(m_j)$. A system provides receiver anonymity iff a passive adversary can determine that $c \in \mathcal{S_A}$ and $receiver(m_j) \in \mathcal{S_A}$, but *not* that $c = receiver(m_j)$ with probability greater than $\frac{1}{|\mathcal{S_A}|} + \epsilon$.

Definition 3. Unlinkability: A system provides unlinkability if it provides both sender and receiver anonymity as no two messages can be attributed to any client.

4.3 Basic Architecture

The simple version of the MIPNet architecture, presented for ease of understanding, works as follows: an Oblivious Proxy OP engages a client C_i in an SFE exchange. As its input, OP provides a mailbox M, which contains all communications currently being exchanged between members of the MIPNet. The mailbox itself contains a number of slots, each consisting of a two bit vector, for each of the n participants. The first bit of the slot, $M_{i,0}$ (the "read bit"), signifies whether or not the vector represents real communication. The second bit of the slot, $M_{i,1}$ (the "data bit"), represents a bit of data being sent to a client. Note that if $M_{i,0} = 0$, the value of $M_{i,1}$ is of no consequence to the receiver. This allows participating clients to inject cover traffic when they are not attempting to communicate with another participating node. C_i inputs its identifier i, at least one vector R_i and the intended destination j of R_i.

Instead of simply using OTs to prevent OP from learning the slot read by a client, we use SFE to gain anonymity of reads and to make the message and slot written by the client indistinguishable from the other messages in the mailbox. We achieve this additional guarantee as follows. All nodes participating in the MIPNet share a keystream k, and enter the current and "next" value of k as an input to the exchange. The keystream is used to blindly decrypt and reencrypt (via XOR within the SFE exchange itself, which is natively supported by Fairplay) all of the slots in the mailbox. Because of the properties of SFE, we can perform these operations without exposing the contents of the remaining $n-1$ slots *not* read by C_i.

As output, C_i receives the decrypted string of bits stored in M_i and OP receives a new set of pseudorandom bits M'. While C_i has blindly changed the *appearance* of all of the slots within the mailbox M, it has only changed the *content* of a single slot (as guaranteed by the circuit). Because all slots appear to have been overwritten from the perspective of OP, C_i is equally as likely to have written to any one of the slots. Accordingly, a passive adversary able to see both M and M' can not guess which message was written by C_i with probability greater than $\frac{1}{n} + \epsilon$.

OP then engages C_{i+1} and continues to service all remaining clients in order based on the ring topology of the MIPNet. This service model removes the ability to infer communications between two clients based on the frequency and temporal relationship of reads and writes.

It is important to note that the assumption of a shared key is not necessarily unrealistic. Even with such an assumption, the inability to read the entire contents of M limits the amount of data a single malicious client can leak per round. However, we remove this assumption in the next subsection to make MIPNets more robust.

4.4 Improved Architecture

The basic MIPNet architecture works well given a number of assumptions. Most importantly, it requires that all clients are well behaved and that they do not attempt to deny service by always transmitting messages to all or a subset of the other clients in each round. Additionally, it assumes that all clients will protect the keystream used to encrypt all communications between members. Such expectations are *not* necessarily realistic given that the anonymity of

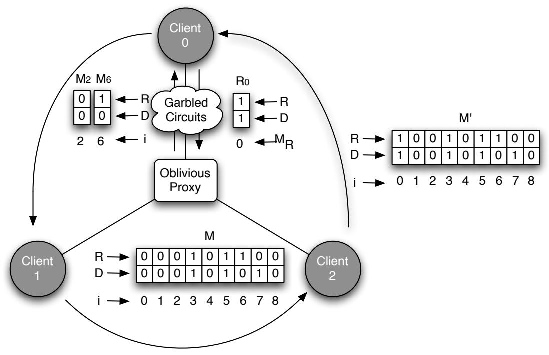


Figure 2: Improved architecture for MIPNets: C_0 and OP exchange messages (encryption removed for example). M' is shown to the side to show the output as seen by OP - note the only change from M occurs in slot 0. When the contents of the mailbox are reencrypted via the SFE exchange, it is not possible for OP to determine which slot has been updated by C_0.

communications in currently deployed systems can be breached using a number of more active techniques [79, 76, 38, 49, 74, 48, 64, 57, 75, 67, 43, 3]. Accordingly, we must make such a system more robust against the real adversaries it is likely to face. We specifically seek to make collusion to link or leak communications and denial of service attacks more difficult for adversaries.

4.4.1 Improved Mechanisms

A compromised client and malicious OP could collaborate to determine the pairs of communicating legitimate clients in the simple MIPNet architecture. The malicious client C_M could give OP access to the keystream k shared between all participating nodes. The OP could then simply decrypt M after each exchange and determine the destination, content and intent (i.e., whether or not the read-bit was set) of the message sent by that client. We address this concern through the use of shared keys between adjacent client nodes. Instead of encrypting M using a k shared between all participants, each client C_i will encrypt the contents of M using the pairwise keystream $k_{C_i,C_{i+1}}$ it shares with the next recipient. This approach makes the leakage of communications far more difficult with an arbitrary compromised client as only C_{i+1} can successfully extract messages from M. We discuss additional mechanisms to reduce the threat of multiple colluding clients in our extended technical report.

A number of techniques to address the denial of service issue have been proposed in superposed anonymous communications systems. Through channel reservation, k-anonymous protocols and a variety of additional techniques [72, 7, 71, 18, 70, 28], such attacks can be reduced. We instead propose to mitigate such attacks through the use of additional storage and a modification to the previous protocol. Specifically, we expand the size of the mailbox M to contain a total of $O(mn)$ bits for a system supporting n users, where m is a coefficient decided upon by the system. Instead of writing to a constant destination, clients select the destination slot based on a secret shared between them. For instance, a client C_i communicates with another client C_j in round S by placing a message in slot $M_R = MAC(S, k_{C_i,C_j})\ mod\ (m \times n)$. During the next exchange, these two clients communicate via $M'_R = MAC(S{+}1, k_{C_i,C_j})\ mod\ (m{\times}n)$. We discuss how m is selected and characterize the probability of collisions in the next subsection.

Like the basic protocol, OP selects M as its input to the exchange. C_i inputs the vector R_i it wishes to send and selects the slot to which the message should be written. Because multiple clients can now legitimately communicate with C_i concurrently, C_i calculates the $n-1$ mailboxes from which it should read using the hashing method described above. C_i also inserts the current and "next" values from the keystream $k_{C_i,C_{i+1}}$. As output, C_i receives $n-1$ slots and OP receives M'. Note that even if a participating adversary were to attempt to deny service by overwriting mailboxes intended for other legitimate nodes, it would be unable to determine the intended destination of the communication because of the randomization of slots within M.

Figure 2 provides an overview of this architecture without the use of encryption to ease understanding. C_0 sends the bit 1 to C_1, signified by the both the Read and Data bits of R_0 being set to this value. C_0 also specifies M_0 as the destination as this was the slot calculated using the previously mentioned technique. OP inserts the entire mailbox M as its input. Through the use of our SFE circuit, C_0 receives M_2 and M_6, which contain a read bit of 0 from C_1 and a read bit 1 and a data bit 0 from C_2, respectively. OP receives M', which differs from M only in one slot (M_0). Note that when encryption is being used, all of the contents of M' will appear to have been changed from the perspective of OP.

We note that the above protocol description does not provide any guarantees of collision detection and retransmission within the architecture itself. Accordingly, we require that such functionality is specified by the application and demonstrate the use of such mechanisms in our extended technical report.

4.4.2 Communication Characteristics

Our "spread-spectrum" inspired mailbox selection approach prevents malicious clients or a compromised OP from regularly overwriting the messages destined for a single node. However, legitimate collisions remain possible. We characterize the probability of such events by further understanding the relationship between the number of clients n, the mailbox multiplication coefficient m and the number of messages s a client sends per exchange with the OP. Because more than one client can send to a single participant in this system without necessarily colliding with other messages, we assume that each client reads $n-1$ messages per exchange with the OP. Additionally, while clients can send s messages per exchange, only one can be sent to a specific participant.

We focus on the probability of collisions during the steady-state operation of the protocol. When a client C_i wants to send messages to s other clients, at most $s(n-1)$ slots will be occupied by unread messages. Each client will therefore have at least $mn - s(n-1)$ open positions to fill with its new messages. Before sending any messages, C_i reads all of the messages sent to it. We can be certain that all s messages sent by C_{i+1} during the previous round have been read. Similarly, we can say that at least $s-1$ messages by C_{i+2} have also been read, and so on. Accordingly, of the $s(n-1)$ slots that have been previously filled, at least $s + (s-1) + (s-$

$2) + \cdots + 1$ messages have already been read. These slots can be treated as empty, making the total number of available positions:

$$A \geq mn - s(n-1) + \frac{s(s+1)}{2}$$

The number of messages that C_i can submit per round must be less than or equal to the number of available slots. Accordingly:

$$\begin{aligned} s &\leq mn - s(n-1) + \frac{s(s+1)}{2} \\ ns - \frac{s(s+1)}{2} &\leq mn \\ s(1 - \frac{s+1}{2n}) &\leq m \end{aligned}$$

Given that clients can submit between 1 and $n-1$ messages per exchange, a system based on this protocol must set a lower bound for m to between $1 - \frac{1}{n}$ and $\frac{n-1}{2}$ to provide the minimum conditions for all s messages to be sent without collision.

We calculate the probability of having no collisions during a round. We assume that there have been no prior collisions in the previous $n-1$ exchanges as this gives us a worst case analysis (i.e., fewer available slots). We let x be equal to the number of slots that are filled when C_i begins its exchange. Specifically:

$$\begin{aligned} x &= s(n-1) - \frac{s(s+1)}{2} \\ &= s((n-1) - \frac{s+1}{2})) \end{aligned}$$

The probability that no collisions occur during a round is therefore:

$$\begin{aligned} P_i &\geq \frac{mn-x}{mn} \times \frac{mn-(x+1)}{mn} \times \cdots \times \frac{mn-(x+s-1)}{mn} \\ &\geq (1 - \frac{x}{mn}) \times (1 - \frac{x+1}{mn}) \times \cdots \times (1 - \frac{x+s-1}{mn}) \\ &\geq \prod_{i=1}^{s} (1 - \frac{x+i-1}{mn}) \end{aligned}$$

This derivation provides an important insight. Ensuring a low probability of collision requires the values $\frac{x}{ms}$ and $\frac{x+k-1}{mn}$ to be as close to 0 as possible. Such conditions are only possible if the value of s is reasonably small (i.e., closer to 1) when compared to n or if the value of m is very large as compared to s.

The probability that an individual client experiences a collision during the course of a round is:

$$P \leq (1 - \frac{s}{nm})^{n-1}$$

Note that the probability of collision for a single node is dependent on the distance (i.e., the number of exchanges between other clients and OP) between the source and destination of a message. We study the performance tradeoffs associated with varying these parameters in the next section.

4.5 Scaling

The "improved" architecture works well for relatively small groups of nodes. However, given the high "end-to-end" delay associated with an increasing number of users and a strict round-robin scheduling algorithm, provisions must be made to aid graceful scaling.

Our architecture can be scaled to support large numbers of users by partitioning nodes into groups - similar approaches have been applied to address security scaling issues in other networks [46]. Each group is treated as if it is its own MIPNet - a mailbox is "passed" between this subset of participants, each of whom read and write messages through the previously described mechanisms. In addition, nodes are able to read from the mailboxes of all of the other groups associated with the MIPNet. Figure 3 provides an overview of this approach. Note that groups are only able to write to their own mailbox; however, they can communicate across groups by reading from all mailboxes. Additionally, in order for a node in one group to be able to read an entry in another group, all nodes in the same position in each group must have access to the same keystream.

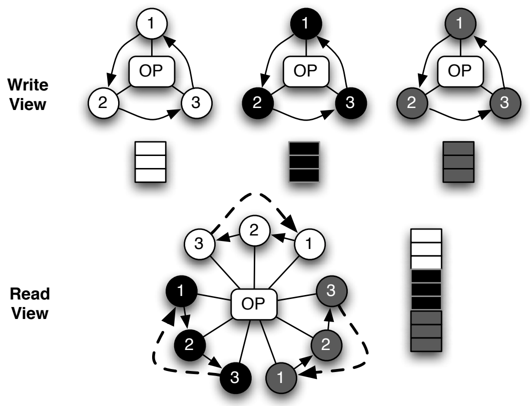


Figure 3: Supporting large numbers of users by separating nodes into groups. Each group of nodes write to their group mailbox (write view). However, nodes in each group are able to read from the other groups' mailboxes (read view). This approach allows operation within the groups to occur in parallel, thereby reducing the end-to-end latency of messages from ng to n, where n is the number of users in each MIPNet and g is the number of MIPNets.

Assuming that we form g MIPNets, each of which contains n nodes for a total of ng users, this approach reduces the time between rounds from ng to n. Messages can accordingly be delivered to their intended destination in much less time.

4.6 Security Guarantees

We now prove that MIPNets provide unlinkability.

LEMMA 1. *MIPNets provide sender anonymity.*

PROOF. The MIPNet architecture is secure against malicious adversaries/attacks based on five assumptions. Like Fairplay, we model SHA-1 as a random oracle and require that a client does not terminate the protocol before sending the output it generates for the OP back to it. Also as in Fairplay, to obtain security against malicious (rather than honest-but curious) adversaries, we rely on a cut-and-choose protocol for distributing garbled circuits to clients. Unlike Fairplay, we implement our OT protocol using a two-lock cryptosystem by Huang et al. [32] based on RSA. Accordingly, each OT operation is secure based on the hardness of the RSA problem. Finally, clients use AES in counter mode to generate a shared keystream, which is pseudorandom assuming that AES is a secure block cipher.

As described in Section 4.4, client C_i inputs the vector R_i, j, i, k, k' into the SFE exchange with OP and receives the slot $i \oplus k$. OP inputs the mailbox M and receives M', which is $M \oplus k \oplus k'$. Because M' and M both appear to be indistinguishable from random bits, it is not possible for OP to guess the slot j written by C_i with probability greater than $\frac{1}{mn} + \epsilon$. As $\frac{1}{mn} \leq \frac{1}{n}$, MIPNets satisfy Definition 1. □

LEMMA 2. *MIPNets provide receiver anonymity.*

PROOF. Receiver anonymity is the dual of sender anonymity as shown in Lemma 1. Accordingly, we rely upon the same cryptographic assumptions. As described in Section 4.4, client C_i receives slot i, but OP is unable to determine the identity of i due to the use of OT with probability greater than $\frac{1}{mn} + \epsilon$. As $\frac{1}{mn} \leq \frac{1}{n}$, MIPNets satisfy Definition 2. $\square$

THEOREM 1. *MIPNets provide unlinkability.*

PROOF. By Lemmas 1 and 2 and Definition 3, MIPNets provide unlinkability. $\square$

5. PERFORMANCE ANALYSIS

In this section, we explore a range of optimizations and tradeoffs in the hopes of providing the most efficient MIPNet architecture.

5.1 Experimental Setup

We analyzed the performance of both the new OT primitive we have implemented and the "improved" MIPNet architecture. Our experiments were run on two servers, each with 8 2-GHz Quad-Core AMD Opteron processors, 16 GB of memory, 1 Gbit Ethernet card and running 2.6.24 Linux Kernel (Ubuntu 8.04.2). The client and OP code were implemented in SFDL and compiled into Java objects using Fairplay.

5.2 Performance Evaluation

5.2.1 OT Performance

We first compare the performance of the two OT primitives. As a simple comparison, we evaluate the performance of each mechanism in the Millionaire's Problem over a range of input sizes. The results are shown in Figure 4 for 100 iterations of the protocol, with 95% confidence intervals included.

As an interesting aside, our first set of results for these experiments exhibited a saw-toothed behavior for a small number of input bits. After much debugging, we realized that the observed irregularities in our timing data were a result of Nagle's Algorithm acting upon TCP. In particular, transfers were being delayed until either buffers were entirely filled or a timeout occurred. We were able to eliminate such behavior by setting the `TCP_NODELAY` socket option, and use this setting throughout the remainder of our experiments.

The results of this experiment clearly show that for the same circuit, the RSA-OT primitive is significantly more efficient than the NP-OT primitive provided with Fairplay. For all of the tested input values, the RSA-OT scheme completed its execution in less than 50% of the time as was required by the NP-OT scheme. Moreover, the use of additional JIT optimization further improved the performance of the RSA-OT scheme by approximately 20%. While the performance difference between the current and new mechanisms on a per-OT scale is small, the use of a relatively large number of OT exchanges makes the aggregate improvement significant. Accordingly, we use the RSA-OT primitive with JIT optimizations as the basis for the remainder of our experiments.

5.2.2 Circuit Size

The size of the garbled circuits created by Fairplay is directly proportional to their performance. However, equivalent but more efficient encodings of these functions are possible. *Ordered Binary Decision Diagrams* (OBDDs) provide a graph-based representation of SFE circuits and have been demonstrated to significantly reduce the bandwidth used by such exchanges. However, in some rare cases, OBDDs can experience an exponential blowup in size and become far less efficient than the circuits produced by Fairplay.

In order to improve the performance of our protocol, we compare the sizes of the functions generated by both Fairplay and the OBDD

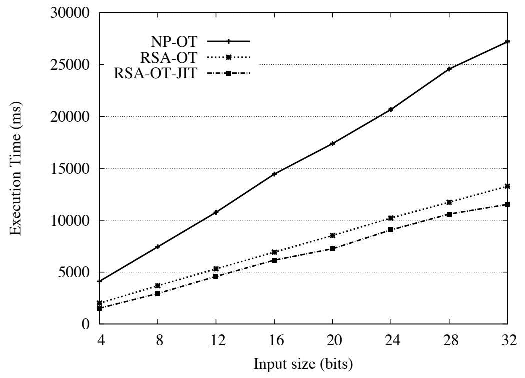


Figure 4: Performance evaluation for the "Millionaire's Problem" comparing the oblivious transfer scheme included with Fairplay (NP-OT) and the RSA-based Two-Lock Cryptosystem (RSA-OT). RSA-OT is more than 50% more efficient than the NP-OT mechanism.

Data	Fairplay			OBDD		
Bits	n=3	n=5	n=7	n=3	n=5	n=7
1	396	3264	6272	292	2328	4590
2	483	3793	7253	380	2856	5562
3	570	4322	8234	468	3384	6534
4	657	4851	9215	556	3912	7506
5	744	5380	10196	644	4440	8478
6	831	5909	11177	732	4968	9450
7	918	6438	12158	820	5496	10422

Table 1: Circuit size for Fairplay and OBDD compiler circuits.

compiler created by Kruger et al. [37]. The results of these tests are provided in Table 1. For a fixed number of users in the MIPNet ($n = 3$, $n = 5$ and $n = 7$) and a fixed $m = n - 1$, we varied the number of data bits carried in each input vector (R) entered by the client per exchange. The functions produced by the OBDD compiler were smaller in all cases, with improvements ranging from 11% to 27% for $n = 3$, 15% to 29% for $n = 5$ and 14% to 27% for $n = 7$.

Accordingly, we use the OBDD compiler to generate the functions used in the remainder of the paper.

5.3 Instant Messaging Application

With an understanding of the performance profile of our system, we now explore the potential for applications to run on top of a MIPNet. We implement a simple Instant Messaging (IM) client, which we call MIPChat. As an example, we envision MIPChat being used in a low cross-traffic scenario to provide a deniable communications medium between a reporter and his or her source as they sit on opposite sides of a public space such as a coffee shop or train station. MIPChat resembles an IRC chat client, with received communications multiplexed into a single window. Outgoing messages are run in a second screen, and prepended with the identifier of the intended destination (e.g., "`@alice`" is mapped to Client 2).

Figure 5 provides an overview of a MIPChat client. Clients without messages to send generate random messages via OpenSSL's `RAND_bytes()` call, which generates cryptographically strong pseudo-random byte streams, and pass these messages to a buffer serviced by the underlying MIPNet client. These messages are sent to an arbitrary client with the read bit in the slot set to zero so as

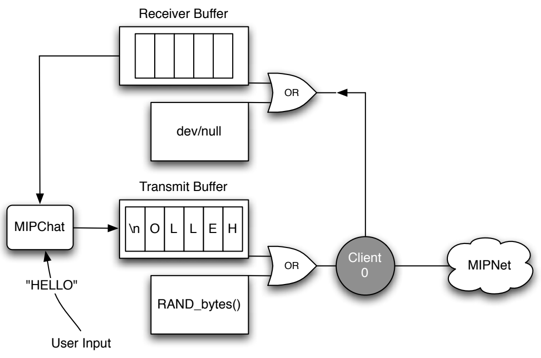
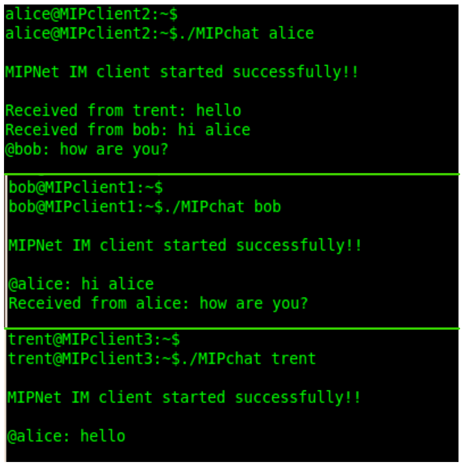


Figure 5: The MIPChat application and its relationship to the MIPNet architecture.



```
alice@MIPclient2:~$
alice@MIPclient2:~$./MIPchat alice

MIPNet IM client started successfully!!

Received from trent: hello
Received from bob: hi alice
@bob: how are you?
```

```
bob@MIPclient1:~$
bob@MIPclient1:~$./MIPchat bob

MIPNet IM client started successfully!!

@alice: hi alice
Received from alice: how are you?
```

```
trent@MIPclient3:~$
trent@MIPclient3:~$./MIPchat trent

MIPNet IM client started successfully!!

@alice: hello
```

Figure 6: A screenshot of the MIPChat client. Clients inject pseudorandom garbage until they have real messages to send. Messages are buffered and displayed by the client when they are received in totality.

to indicate the lack of content in the message. The receiving client simply discards the messages and awaits the next exchange. When the user presses return, the real message is parsed for destination and sent to the same buffer, where it is sent to the requested host. On the receiving side, the client collects incoming legitimate messages in a buffer and, upon receiving an `EOL` character, pushes the contents of the buffer to the client where they are displayed for the user. Figure 6 provides a screen shot of this application, in which "Bob" and "Trent" have both sent messages to "Alice", and Alice has responded to "Bob". Recall that the source of the messages is clear to the receiver in the absence of collisions as only a specific sequence of slots in the mailbox could be correctly filled by the sender based on the shared key hashing mechanism discussed in Section 4.4.

Two small changes to the underlying MIPNet can allow for the efficient support of a much larger user population. First, we break users into groups as was proposed in Section 4.5 to reduce the "end-to-end" latency caused by adding users. Second, we take advantage of how users are likely to actually use the MIPChat client. Specifically, users are unlikely to attempt to talk to every participant in the network concurrently. While a user may exchange messages with a small number of participants at the same time, few users will ever talk to every member on their contact list in parallel. From an informal analysis of chat logs in our own lab, we saw that no user

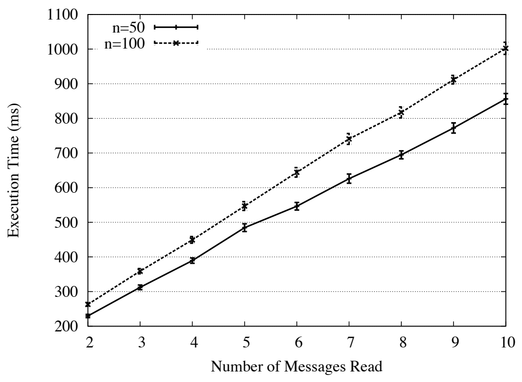


Figure 7: The performance profile of MIPChat running on top of our architecture. Note that we can support large user populations (e.g., $ng = 100$) with this approach and still process keystrokes as or nearly as fast as users enter them.

ever spoke to more than three individuals on their contact list at the same time. The execution time of an SFE exchange with an OP can therefore be dramatically reduced by limiting the number of slots actually read by a client in a single interaction. Users can check the slots in which they expect a message during each round and also intermittently monitor other slots in the mailbox for attempts to initiate communications. We leave the specific synchronization scheme between clients (e.g., explicit "start" messages, synchronized sleep periods, etc) to each implementation of this system as these mechanisms have been thoroughly studied.

Figure 7 shows the performance of the MIPChat client supporting $ng = 50, 100$ clients, $m = 2$, $s = 1$, 5 nodes assigned to each group and a varying number of the total slots read. As before, this experiment shows the mean of 100 iterations of the experiment for each point and provides 95% confidence intervals two orders of magnitude smaller than the mean. This approach provides significant savings over the standard MIPNet architecture. Specifically, if a client only reads five messages during each round (e.g., three conversations and polls for two additional conversation requests), each round takes an average of 484.43 and 546.32 ms for $ng = 50, 100$, respectively. Because users are not constantly typing, the slightly slower processing to input speed here is unlikely to significantly impact perception of throughput for this application. Note the longest delay to deliver a message (i.e., from client i to client $i - 1$) has also been greatly reduced in this configuration to approximately three seconds. Given that the average user types between 19 and 33 words per minute [36], or one character every 333.3 to 526.3ms, *our infrastructure is capable of processing characters faster than the majority of users enter them for all but one of the above cases.*

These values indicate that the MIPNet architecture is capable of supporting applications with some tolerance to delay (e.g., chat). We will show the practicality of additional applications including email in our future work.

5.4 Performance Comparison

We conduct one final set of experiments to measure the total performance improvement over the naïve use of the underlying cryptographic primitives. Specifically, we recreate the MIPChat client without any of the optimizations discussed in this work (e.g., RSA-OT, groups, etc) and simply compile our circuits using an unmodified version of the Fairplay compiler and execute them in the "basic" architecture.

Unfortunately, this most basic comparison was not possible because the Fairplay compiler exceeded the 1GB memory maximum set by the Java virtual machine. Accordingly, we compiled our circuits using only the OBDD compiler but without any additional optimizations. For $n = 100$, we recorded 50 iterations of the protocol and observed an average time of 35.626 seconds, with a 95% confidence interval of ± 0.161 seconds. The performance profile of our optimized MIPChat client provides a reduction in execution time of 98.5%. This dramatic improvement over the naïve application of SFE primitives demonstrates that our optimizations and careful parameterization can allow these constructions to form the basis of a system with near real-time performance requirements.

6. CONCLUSION

We present Mix-In-Place Networks (MIPNets), an architecture built on Secure Function Evaluation that replaces multiple intermediary nodes with a cascade of functions in a single proxy. Through the use of SFE, this proxy remains oblivious to the source, destination and content of all messages exchanged within the system. We demonstrate that our proposed architecture provides provable deniability and then, through extensive performance analysis using a number of optimizations that dramatically reduce the execution time (greater than 98%), demonstrate the practicality of this approach for applications with near real-time requirements, including instant messaging for as many as 100 clients. Finally, we discuss a number of enhancements to potentially further improve the performance, scalability and robustness of our architecture. In so doing, we have not only demonstrated the practicality of using a single node to mix traffic with provable properties, but also shown that SFE can address a void in the anonymous communications space for applications with moderate performance constraints and no guarantees of adequate cross traffic.

Acknowledgments

We would like to thank Louis Kruger for providing us with the source code for the OBDD compiler, and Chris Peikert, William Enck, Chaitrali Amrutkar, Brendan Dolan-Gavit and our anonymous reviewers for their helpful comments. This work was supported in part by the US National Science Foundation (CCF-0916031). Any opinions, findings, conclusions or recommendations expressed in this publication are those of the authors and do not necessarily reflect the views of the National Science Foundation.

7. REFERENCES

[1] G. Aggarwal, N. Mishra, and B. Pinkas. Secure Computation of the k^{th}-Ranked Element. In *Proceedings of Eurocrypt*, 2004.

[2] P. Baran. On Distributed Communications: IX. Security, Secrecy, and Tamper-Free Considerations. Technical Report RM-3765-PR, The RAND Corporation, 1964.

[3] K. Bauer, D. McCoy, D. Grunwald, T. Kohno, and D. Sicker. Low-Resource Routing Attacks Against Tor. In *Proceedings of the ACM Workshop on Privacy in Electronic Society (WPES)*, 2007.

[4] A. Beimel and S. Dolev. Buses for Anonymous Message Delivery. *Journal of Cryptology*, 16(1), 2001.

[5] A. Ben-David, N. Nisan, and B. Pinkas. FairplayMP – A System for Secure Multi-Party Computation. In *Proceedings of the ACM Conference on Computer and Communications Security (CCS)*, 2008.

[6] O. Berthold, H. Federrath, and S. Kopsell. Web MIXes: A System for Anonymous and Unobservable Internet Access. In *Proceedings of Designing Privacy Enhancing Technologies*, 2000.

[7] J. Bos and B. den Boer. Detection of Disrupters in the DC Protocol. In *Proceedings of Eurocrypt*, 1989.

[8] J. Boyan. The Anonymizer: Protecting User Privacy on the Web. *Computer-Mediated Communication Magazine*, 4(9), 1997.

[9] M. Burnside and A. Keromytis. Low Latency Anonymity with Mix Rings. In *Proceedings of the 9th International Information Security Conference (ISC)*, 2006.

[10] D. Chaum. Untraceable electronic mail, return addresses and digital pseudonyms. *Communications of the ACM*, 24(2), 1981.

[11] D. Chaum. The Dining Cryptographers Problem: Unconditional Sender and Recipient Untraceability. *Journal of Cryptology*, 1(1), 1988.

[12] I. Clarke, S. G. Miller, T. W. Hong, O. Sandberg, and B. Wiley. Protecting Free Expression Online with Freenet. *IEEE Internet Computing*, January/February 2002.

[13] C. Cortes, D. Pregibon, and C. Volinsky. Communities of Interest. In *Proceedings of the International Symposium of Intelligent Data Analysis (IDA)*, 2001.

[14] W. Dai. PipeNet 1.1. `http://www.weidai.com`, 1998.

[15] G. Danezis, R. Dingledine, and N. Mathewson. Mixminion: Design of a Type III Anonymous Remailer Protocol. In *Proceedings of the IEEE Symposium on Security and Privacy (OAKLAND)*, 2003.

[16] R. Dingledine, M. J. Freedman, and D. Molnar. The Free Haven Project: Distributed Anonymous Storage Service. In *Proceedings of the Workshop on Design Issues in Anonymity and Unobservability*, 2000.

[17] R. Dingledine, N. Mathewson, and P. Syverson. Tor: The Second-Generation Onion Router. In *Proceedings of the USENIX Security Symposium (USENIX)*, 2004.

[18] S. Dolev and R. Ostrovsky. XOR-Trees for Efficient Anonymous Multicast Reception. In *Proceedings of CRYPTO*, 1997.

[19] N. S. Evans, R. Dingledine, and C. Grothoff. A Practical Congestion Attack on Tor Using Long Paths. In *Proceedings of the USENIX Security Symposium (USENIX)*, 2009.

[20] H. Federrath. JAP – Anonymity and Privacy. `http://anon.inf.tu-dresden.de/index_en.html`, 2008.

[21] J. Feigenbaum, A. Johnson, and P. Syverson. A Model of Onion Routing with Provable Anonymity. In *Proceedings of Financial Cryptography*, 2007.

[22] J. Feigenbaum, B. Pinkas, R. Ryger, and F. Saint-Jean. Secure Computation of Surveys. In *Proceedings of the EU Workshop on Secure Multiparty Protocols (SMP)*, 2004.

[23] M. Freedman, K. Nissim, and B. Pinkas. Efficient Private Matching and Set Intersection. In *Proceedings of Eurocrypt*, 2004.

[24] M. J. Freedman and R. Morris. Tarzan: A Peer-to-Peer Anonymizing Network Layer. In *Proceedings of the ACM Conference on Computer and Communications Security (CCS)*, 2002.

[25] I. Goldberg and A. Shostack. Freedom Systems 2.0 Architecture. `http://osiris.978.org/%7Ebrianr/crypto-research/anon/www.freedom.net/products/whitepapers/Freedom_System_2_Architecture.pdf`, 1999.

[26] O. Goldreich, S. Micali, and A. Wigderson. How to Play Any Mental Game or A Completeness Theorem for Protocols with Honest Majority. In *Proceedings of the ACM Symposium on Theory of Computing (STOC)*, 1987.

[27] P. Golle, M. Jakobsson, A. Juels, and P. Syverson. Universal re-encryption for mixnets. In *Proceedings of the RSA Conference, CryptographerâĂŹs Track*, 2004.

[28] P. Golle and A. Juels. Dining Cryptographers Revisited. In *Proceedings of Eurocrypt*, 2004.

[29] M. Green and G. Ateniese. Identity-Based Proxy Re-encryption. In *Applied Cryptography and Network Security (ACNS)*, 2007.

[30] C. Gulcu and G. Tsudik. Mixing Email with Babel. In *Proceedings of the ISOC Symposium on Network and Distributed Systems Security (NDSS)*, 1996.

[31] T. Heydt-Benjamin, A. Serjantov, and B. Defend. Nonesuch: A Mix Network with Sender Unobservability. In *Proceedings of the 5th ACM workshop on Privacy in electronic society*, 2006.

[32] H.-F. Huang and C.-C. Chang. A New Design for Efficient t-out-n Oblivious Transfer Scheme. In *Proceedings of the International Conference on Advanced Information Networking and Applications (AINA)*, 2005.

[33] Y. Huang, D. Evans, J. Katz, and L. Malka. Faster Secure Two-Party Computation Using Garbled Circuits. In *Proceedings of the USENIX Security Symposium (SECURITY)*, 2011.

[34] M. Jakobsson. Flash Mixing. In *Proceedings of the ACM Principles of Distributed Computing (PODC)*, 1999.

[35] L. Johansen, M. Rowell, K. Butler, and P. McDaniel. Email Communities of Interest. In *Proceedings of the Conference on Email and Anti-Spam (CEAS)*, 2007.

[36] C. M. Karat, C. Halverson, D. Horn, and J. Karat. Patterns of Entry and Correction in Large Vocabulary Continuous Speech Recognition Systems. In *Proceedings of the ACM SIGCHI Conference on Human Factors in Computing Systems (CHI)*, 1999.

[37] L. Kruger, S. Jha, E.-J. Goh, and D. Boneh. Secure Function Evaluation with Ordered Binary Decision Diagrams. In *Proceedings of the ACM Conference on Computer and Communications Security (CCS)*, 2006.

[38] B. N. Levine, M. K. Reiter, C. Wang, and M. K. Wright. Timing Attacks in Low-Latency Mix-based Systems. In *Proceedings of Financial Cryptography*,

2004.

[39] B. N. Levine and C. Shields. Hordes: A Multicast Based Protocol for Anonymity. *Journal of Computer Security*, 10, 2002.

[40] Y. Lindell and B. Pinkas. Privacy preserving data mining. 15(3), 2002.

[41] P. D. MacKenzie, A. Oprea, and M. Reiter. Automatic Generation of Two-Party Computations. In *Proceedings of the ACM Conference on Computer and Communications Security (CCS)*, 2003.

[42] D. Malkhi, N. Nisan, B. Pinkas, and Y. Sella. Fairplay - A Secure Two-Party Computation System. In *Proceedings of the USENIX Security Symposium (SECURITY)*, 2004.

[43] N. Mathewson and R. Dingledine. Practical Traffic Analysis: Extending and Resisting Statistical Disclosure. In *Proceedings of the Privacy Enhancing Technologies Symposium (PETS)*, 2005.

[44] P. McDaniel, S. Sen, O. Spatscheck, J. Van der Merwe, B. Aiello, and C. Kalmanek. Enterprise Security: A Community of Interest Based Approach. In *Proceedings of the Symposium on Network and Distributed System Security (NDSS)*, 2006.

[45] C. A. Melchor and Y. Deswarte. From DC-nets to pMIXes: Multiple Variants for Anonymous Communications. In *Proceedings of the IEEE Symposium on Network Computing and Applications (NCA)*, 2006.

[46] S. Mittra. Iolus: A Framework for Scalable Secure Multicasting. In *Proceedings of ACM SIGCOMM*, 1997.

[47] U. Moeller, L. Cottrell, P. Palfrader, and L. Sassaman. Mix-master Protocol Version 2. Technical Report Internet-Draft, IETF Network Working Group, 2004.

[48] S. J. Murdoch. Hot or Not: Revealing Hidden Services by their Clock Skew. In *Proceedings of the ACM Conference on Computer and Communications Security (CCS)*, 2006.

[49] S. J. Murdoch and G. Danezis. Low-cost Traffic Analysis of Tor. In *Proceedings of the IEEE Symposium on Security and Privacy (OAKLAND)*, 2005.

[50] M. Naor and B. Pinkas. Efficient Oblivious Transfer Protocols. In *Proceedings of the ACM Symposium on Discrete Algorithms (SODA)*, 2001.

[51] A. Pfitzmann, B. Pfitzmann, and M. Waidner. ISDN-mixes: Untraceable Communication with Very Small Bandwidth Overhead. In *Proceedings of the GI/ITG Conference on Communication in Distributed Systems*, 1991.

[52] A. Pfitzmann and M. Waidner. Networks Without User Observability. *Computers & Security*, 6, 1987.

[53] B. Pinkas, M. Naor, and R. Sumner. Privacy Preserving Auctions and Mechanism Design. In *Proceedings of the ACM Conference on Electronic Commerce*, 1999.

[54] K. Poulsen. FBI retires its Carnivore. `http://www.securityfocus.com/news/10307`, 2005.

[55] J. Powers. The attack of the eavesdropping neighbors. `http://www.michigandaily.com/content/attack-eavesdropping-neighbors`, 2002.

[56] Proxify.com. Proxify anonymous proxy - surf the Web privately and securely. `http://proxify.com/`, 2008.

[57] Y. J. Pyun, Y. H. Park, X. Wang, D. Reeves, and P. Ning. Tracing Traffic Through Intermediate Hosts that Repacketize Flows. In *Proceedings of IEEE INFOCOM*, 2007.

[58] M. O. Rabin. How to Exchange Secrets with Oblivious Transfer. Technical Report TR-81, Aiken Computation Lab, Harvard University, 1981.

[59] M. Reed, P. Syverson, and D. Goldschlag. Proxies for Anonymous Routing. In *Proceedings of Annual Computer Security Applications Conference (ACSAC)*, 2006.

[60] C. Reis, S. D. Gribble, T. Kohno, and N. Weaver. UW CSE and ICSI Web Integrity Checker. `http://vancouver.cs.washington.edu/`, 2007.

[61] M. Reiter and A. Rubin. Crowds: Anonymity for Web Transactions. *ACM Transactions on Information System Security (TISSEC)*, 1(1), 1998.

[62] L. Sassaman, B. Cohen, and N. Mathewson. The Pynchon Gate: A Secure Method of Pseudonymous Mail Retrieval. In *Workshop on Privacy in the Electronic Society (WPES)*, 2005.

[63] R. Sherwood, B. Bhattacharjee, and A. Srinivasan. P^5: A Protocol for Scalable Anonymous Communication. In *Proceedings of the IEEE Symposium on Security and Privacy (OAKLAND)*, 2002.

[64] V. Shmatikov and M.-H. Wang. Timing Analysis in Low-Latency Mix Networks: Attacks and Defenses. In *Proceedings of the European Symposium on Research in Computer Security (ESORICS)*, 2006.

[65] R. Singel. Google-DoubleClick Privacy Fight Hangs Over Fed's E-Advertising Forum. `http://blog.wired.com/27bstroke6/2007/10/google-doublecl.html`, 2007.

[66] E. G. Sirer, S. Goel, M. Robson, and D. Engin. Eluding Carnivores: File Sharing with Strong Anonymity. In *Proceedings of the European SIGOPS Workshop*, 2004.

[67] M. Srivatsa, L. Liu, and A. Iyengar. Preserving Caller Anonymity in Voice-over-IP Networks. In *Proceedings of the IEEE Symposium on Security and Privacy (OAKLAND)*, 2008.

[68] P. Syverson, D. Goldschlag, and M. Reed. Anonymous Connections and Onion Routing. In *Proceedings of the IEEE Symposium on Security and Privacy (OAKLAND)*, 1997.

[69] P. Syverson, G. Tsudik, M. Reed, and C. Landwehr. Towards an Analysis of Onion Routing Security. In *Workshop on Design Issues in Anonymity and Unobservability*, 2000.

[70] L. von Ahn, A. Bortz, and N. J. Hopper. k-Anonymous Message Transmission. In *Proceedings of the ACM Conference on Computer and Communications Security (CCS)*, 2003.

[71] M. Waidner. Unconditional Sender and Recipient Untraceability in Spite of Active Attacks. In *Proceedings of Eurocrypt*, 1990.

[72] M. Waidner and B. Pfitzmann. The Dining Cryptographers in the Disco: Unconditional Sender and Recipient Untraceability with Computationally Secure Service ability. In *Proceedings of Eurocrypt*, 1989.

[73] M. Waldman, A. D. Rubin, and L. F. Cranor. Publius: A robust, tamper-evident, censorship-resistant web publishing system. In *Proceedings of the USENIX Security Symposium (SECURITY)*, 2000.

[74] X. Wang, S. Chen, and S. Jajodia. Tracking Anonymous Peer-to-Peer VoIP Calls on the Internet. In *Proceedings of the ACM Conference on Computer and Communications Security (CCS)*, 2005.

[75] X. Wang, S. Chen, and S. Jajodia. Network Flow Watermarking Attack on Low-Latency Anonymous Communication Systems. In *Proceedings of the IEEE Symposium on Security and Privacy (OAKLAND)*, 2007.

[76] X. Wang, D. Reeves, and S. Wu. Inter-packet Delay Based Correlation for Tracing Encrypted Connections Through Stepping Stones. In *Proceedings of the European Symposium on Research in Computer Security (ESORICS)*, 2002.

[77] A. C. Yao. How to Generate and Exchange Secrets. In *Proceedings of the IEEE Symposium on Foundations of Computer Science (FOCS)*, 1986.

[78] YouHide.com. Anonymous Proxy Server. `http://www.youhide.com/`, 2008.

[79] Y. Zhang and V. Paxson. Detecting Stepping Stones. In *Proceedings of the USENIX Security Symposium (SECURITY)*, 2000.

Security Through Amnesia: A Software-Based Solution to the Cold Boot Attack on Disk Encryption

Patrick Simmons
University of Illinois at Urbana-Champaign
simmon12@illinois.edu

ABSTRACT

Disk encryption has become an important security measure for a multitude of clients, including governments, corporations, activists, security-conscious professionals, and privacy-conscious individuals. Unfortunately, recent research has discovered an effective side channel attack against any disk mounted by a running machine [23]. This attack, known as the cold boot attack, is effective against any mounted volume using state-of-the-art disk encryption, is relatively simple to perform for an attacker with even rudimentary technical knowledge and training, and is applicable to exactly the scenario against which disk encryption is primarily supposed to defend: an adversary with physical access.

While there has been some previous work in defending against this attack [27], the only currently available solution suffers from the twin problems of disabling access to the SSE registers and supporting only a single encrypted volume, hindering its usefulness for such common encryption scenarios as data and swap partitions encrypted with different keys (the swap key being a randomly generated throw-away key). We present Loop-Amnesia, a kernel-based disk encryption mechanism implementing a novel technique to eliminate vulnerability to the cold boot attack. We contribute a novel technique for shielding multiple encryption keys from RAM and a mechanism for storing encryption keys inside the CPU that does not interfere with the use of SSE. We offer theoretical justification of Loop-Amnesia's invulnerability to the attack, verify that our implementation is not vulnerable in practice, and present measurements showing our impact on I/O accesses to the encrypted disk is limited to a slowdown of approximately 2x. Loop-Amnesia is written for x86-64, but our technique is applicable to other register-based architectures. We base our work on loop-AES, a state-of-the-art open source disk encryption package for Linux.

1. INTRODUCTION

The theft of sensitive data from computers owned by governments, corporations, and legal and medical professionals has escalated to a problem of paramount importance as computers are now used to store, modify, and safeguard all kinds of sensitive and private information. Hard drive thefts in the past have put information such as medical data [5], Social Security and passport numbers [19], and the access codes for a financial service corporation's private Intranet [3][1] at risk.

Because of the significant potential for harm such breaches represent, disk encryption, in which an entire filesystem is stored on nonvolatile storage in encrypted form, has become a standard and often mandatory security technique in many environments [9]. Most major commercial operating systems now offer some form of kernel-based disk encryption [33] [30] [12], and third-party tools supporting disk encryption, such as TrueCrypt [2], are freely available for many architectures and operating systems. This software has proven effective against determined adversaries wishing to defeat its protection [14].

However, recent work [23] by Halderman et al. has uncovered a flaw common to all commercially available disk encryption packages. These researchers observed that, as long as an encrypted volume is mounted, a disk encryption package will store the encryption key in RAM. They further discovered that, contrary to popular belief, DRAM does not lose its contents for several minutes after a loss of power. Thus, Halderman et al. put forth the following attack on all disk encryption: cut power to the target machine, pull out the RAM, put the RAM in a new machine[2], and boot this machine with an attack program of their creation which overwrites a minimal amount of RAM with its own code and dumps the original contents of RAM to nonvolatile storage. At this point, an attacker can search the contents of RAM for the encryption key or simply try every key-length string of bits present in the RAM of the original machine as a potential key. This attack, called the "cold-boot attack" by Halderman et al., is simple to perform, routinely effective, and broadly applicable against the existing universe of disk encryption software packages.

It is difficult to overstate the significance of the cold-boot attack. The protection afforded by disk encryption against any adversary with access to the running target machine is now effectively skewered. Many users for whom disk encryption previously offered protection are now at risk of having their data stolen when their machines are stolen or lost. One

[1] In this case, the hard drive appears to have been sold and purchased legitimately but was not adequately wiped prior to the sale.

[2] Variants of the attack eliminate the need for a separate machine.

may argue that these users should physically secure their machines, but, as disk encryption is specifically intended to protect against an attacker who has physical access to the disk, that argument rings hollow.

In this paper, we describe the novel implementation approach we used in Loop-Amnesia, one of the first disk encryption software packages not vulnerable to the cold-boot attack. Relative to independently developed concurrent work [27], we contribute a method of permanently storing an encryption key inside CPU registers rather than in RAM without interfering with the use of SSE and an approach of capitalizing on the ability to store a single encryption key to allow the masking of arbitrarily many encryption keys from disclosure under a cold-boot attack. We also contribute a prototype implementation of our approach and providing performance measurements validating our technique's usability in practice.

Section 2 describes the attack model used by our paper. Section 3 provides an overview of AES and the loop-AES software package we enhanced to thwart the cold-boot attack. Section 4 describes the design of Loop-Amnesia. Section 5 describes our implementation. Section 6 describes our justification that Loop-Amnesia is in fact immune to the cold-boot attack and describes our correctness testing. Section 7 details our performance benchmarking of Loop-Amnesia. Section 8 details the limitations of our approach to this problem. Section 9 describes related work. Section 10 describes future work. Section 11 concludes the paper.

2. ATTACK MODEL

We assume our attacker has full physical access to the target machine. The attacker is assumed to possess any commonly available equipment necessary or useful for performing the cold-boot attack, such as his own computer or other device capable of reading RAM after he has removed it from the target machine.

In the event our attacker has access to an account on the target machine, such as with stolen login credentials or due to the fact that the machine was stolen with a user logged in, we seek to prevent the attacker from gaining unauthorized access to the disk volume key or to parts of the encrypted disk to which the account he is using does not have access. We assume an attacker will not be able to gain access to the encryption keys through vulnerabilities in the operating system; other work (SVA [13], SECVisor [36], and HyperSafe [39]) can protect the kernel from exploitation.

3. BACKGROUND

3.1 Aspects of AES Relevant to Loop-Amnesia

AES, or the Advanced Encryption Standard, is an efficient block cipher algorithm. Originally published as Rijndael [15], the algorithm became the AES standard in 2001. It has proven quite resistant to cryptanalysis [8] [20] since its standardization.

3.1.1 Rounds

AES encryption proceeds in multiple *rounds*. In a round-based encryption process, plaintext is first encoded to ciphertext by applying the main body of the encryption algorithm. The resulting ciphertext is then encrypted again using the same algorithm in a second round of encryption. This process is repeated a number of times: in the case of 128-bit AES, our algorithm of concern, the number of rounds is 10.

3.1.2 Key Schedule

In order to increase the algorithm's resistance to cryptanalysis, AES and other block ciphers employ a concept called a *key schedule*, in which a different key is used for each round of encryption. In AES, the original key is used for the first round, and subsequent rounds use keys obtained by permuting the contents of the previous round key. This permutation is reversible. In most AES implementations, all 10 keys of the key schedule are precomputed and stored to RAM for performance purposes.[3] Since there are different but related key schedules for encryption and decryption, a total of 20 128-bit quantities from which the original key can be derived are stored to RAM when using unmodified loop-AES or a similar disk encryption package.

3.2 Organization of loop-AES

Loop-AES [31] is a kernel plugin for Linux providing an *encrypted loopback device* to the user. An encrypted loopback device binds to a normal block device, such as a disk partition or file, and provides a view of its data after having been decrypted with a key. If data is written to the loopback device, it is encrypted before being stored on the device to which the loopback device is bound.

The internal structure of loop-AES is both clean and modular. All encryption, decryption, and key-setting work is performed by the three methods `aes_encrypt`, `aes_decrypt`, and `aes_set_key`. Key data is stored inside the `aes_context` structure, which is treated as opaque by all of loop-AES outside of the three routines mentioned above. IV computation, CBC chaining, and other functions necessary to a full disk encryption system are handled independently of the implementation of these functions and, indeed, independently of the cryptographic algorithm used. Loop-Amnesia's changes to loop-AES are confined to these three subroutines.

Of particular concern to us is how loop-AES stores cryptographic keys. Keys are stored only inside the aforementioned opaque `aes_context` structures; loop-AES conscientiously deletes them from other locations in memory after initializing the `aes_context` structures with `aes_set_key`. Because the keys are stored in memory by `aes_set_key`, however, loop-AES, like other prior disk encryption software, is fully vulnerable to the cold-boot attack.

4. THE DESIGN OF LOOP-AMNESIA

The basic insight of Loop-Amnesia's design is that, because of the ubiquity of model-specific registers, or MSRs, in CPU architectures today, it is possible to store data inside the CPU, rather than in RAM, thus making that data unreadable to a perpetrator of the cold-boot attack. The challenging aspect of this approach is finding model-specific registers that can practicably be used for this task: if an MSR is repurposed as storage space for an encryption key, it is unavailable for its intended use. Model-specific registers are used for a diverse variety of system tasks; some, like the control for the CPU fan, must not be tampered with lightly lest the safe operation of the hardware be threatened.

[3]For reasons discussed in later sections, this performance optimization is foreclosed to Loop-Amnesia.

On our target platform, x86-64, we disabled performance counting and therefore were able to use the performance counter registers to hold a single 128-bit AES key.[4] To evaluate the generality of our approach, we examined the CPU system programming manual for a PowerPC chip [35]. We were also able to find performance counter MSRs on PowerPC that would appear to be repurposable for key storage on that architecture.[5] Of course, on any platform, disabling and repurposing the hardware performance counter infrastructure in this manner has the side effect of foreclosing the use of any hardware-assisted performance profilers. Since we expect protection against cold-boot attacks to be most important for production machines, which do not typically use hardware-assisted performance profilers, we do not consider this a serious deficiency of our approach.

Since storing the disk volume key in the MSRs directly would prevent the mounting of more than one encrypted volume simultaneously[6], we instead store a randomly generated number in the MSRs, then use this master key to encrypt the disk volume key for each mounted volume. Because we assume an attacker may later have access to all RAM, we require a random number generator (RNG) which guarantees that previously output random numbers cannot be calculated from its subsequent internal state.[7]

5. IMPLEMENTATION

5.1 Constraints

To validate our design, we built a cold-boot immune 128-bit AES implementation as a drop-in replacement for the 128-bit AES implementation already present in the loop-AES disk encryption package. In order to satisfy our primary design criterion of cold-boot immunity, we must take care in our implementations of `aes_encrypt` and `aes_decrypt` that no key data is ever stored to RAM. This places a number of constraints on our implementation.

First, in order to ensure no register containing key data could ever be spilled to RAM, we needed a degree of control over the register allocation process not available to the programmer in any high-level language, including C. For this reason, our implementation of Loop-Amnesia uses x86-64 assembly language exclusively.

```
aes_encrypt(context,plaintext_buffer,
        ciphertext_buffer):
- Disable interrupts.
- Read master key from MSRs to registers.
- Read encrypted volume key from memory
    to registers.
- Decrypt volume key without storing
    any temporary data in RAM.
- Read plaintext_buffer from RAM to
    registers.
- Encrypt plaintext using volume key
    without storing any temporary
    data in RAM.
- Write ciphertext to ciphertext_buffer.
- Zero all registers containing key data.
- Enable Interrupts.
- Return
```

```
aes_decrypt(context,ciphertext_buffer,
        plaintext_buffer):
- Disable interrupts.
- Read master key from MSRs to registers.
- Read encrypted volume key from memory
    to registers.
- Decrypt volume key without storing any
    temporary data in RAM.
- Read ciphertext_buffer from RAM to
    registers.
- Decrypt ciphertext using volume key
    without storing any temporary
    data in RAM.
- Write plaintext to plaintext_buffer.
- Zero all registers containing key data.
- Enable Interrupts.
- Return
```

```
aes_set_key(context,key_bytes):
- if this is the first call to aes_set_key:
    master_key = gen_random_bytes(); msr_store(master_key)
- master_key = msr_load()
- first_round_key = internal_decrypt(master_key,key_bytes)
- context->first_round_key = first_round_key
- last_round_key = lastround(key_bytes)
- last_round_key = internal_decrypt(master_key,last_round_key)
- context->last_round_key = last_round_key
```

Figure 1: Pseudocode Description of Loop-Amnesia

Second, though most AES implementations, in order to improve performance, precompute the AES key schedule and cache it to RAM, our repurposed MSR space is far too limited to store even one full AES key schedule. We instead compute the key schedule on-the-fly during encryption and decryption as discussed in §5.2.

Finally, as MSRs are per-CPU (or per-core), the need to copy our master key to all CPUs that may run the Loop-Amnesia subroutines presents a logistical problem. Our prototype implementation currently handles this problem by compiling the Linux kernel in single-CPU mode, forcing all software to execute on only one CPU or CPU core. While the prototype implementation of loop-AES therefore currently limits a machine to a single core, there is nothing in the design of Loop-Amnesia requiring this limitation. In a production implementation of Loop-Amnesia, we would suggest storing the master key to RAM after its generation, forcing all CPUs to read it and store it to their MSRs, and subsequently scrubbing the key from RAM.

The TPM Alternative

Many of these design constraints could be lifted if hardware support were available. However, the Trusted Protection Modules [38] present on so many computers today do not provide useful hardware support for our goal. While it might at first appear that we could secure the key inside of such a cryptographic coprocessor and use it to perform all encryption and decryption of the disk, the current TPM standard only supports the public-key RSA algorithm, which is inappropriate for disk encryption.

However, even though TPMs are not useful for performing the actual disk encryption, they could be used as an alternative method of encrypting the disk volume keys: instead of using an AES key hidden in an MSR on the main processor for the master key, we could use a public RSA key generated by the TPM. When we wanted to perform disk encryption or decryption, we could ask the TPM to use the corresponding private RSA key to decrypt the values we stored in RAM, reading the decrypted disk volume key directly from the TPM to registers over the serial bus.

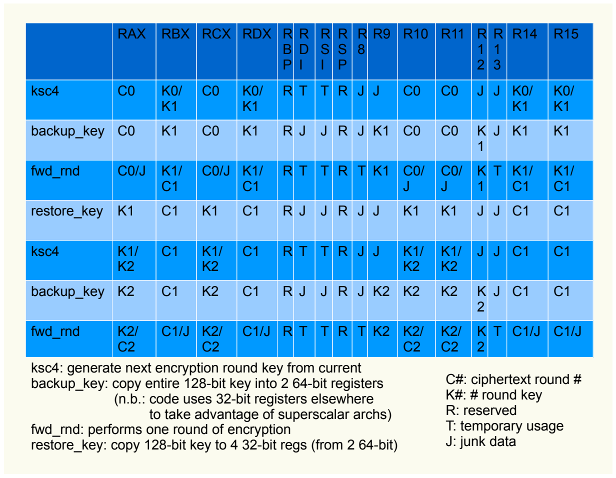
Unfortunately, this is an inferior alternative to our approach from both security and performance standpoints.

[4]On Intel [11] processors, we use MSRs 0xC1, 0xC2, 0x309, and 0x30A. On AMD [16] CPUs, we use MSRs 0xC0010004, 0xC0010005, 0xC0010006, and 0xC0010007.

[5]However, the manual also states that the performance counters are readable from user mode, and it does not appear that the instruction to read them can be disabled by the operating system. Thus, our approach may not provide security against an attacker with the ability to execute arbitrary user-level code on PowerPC unless we found other repurposable MSRs. On x86-64, the ability of unprivileged code to read performance counters is configurable by the operating system, and we disable this ability.

[6]Another motivation for supporting multiple simultaneous encryption keys is to support a mode of loop-AES which uses 64 different encryption keys to protect against watermark attacks [31]

[7]In our implementation, we use the Linux kernel random number generator, which is specifically designed to provide this guarantee. There has been some cryptanalysis of the Linux RNG with respect to its ability to provide this guarantee [22], but the implementation is still considered safe in practice [18].



	RAX	RBX	RCX	RDX	RBP	RDI	RSI	RSP	R8	R9	R10	R11	R12	R13	R14	R15
ksc4	C0	K0/K1	C0	K0/K1	R	T	T	R	J	J	C0	C0	J	J	K0/K1	K0/K1
backup_key	C0	K1	C0	K1	R	J	J	R	J	K1	C0	C0	K1	J	K1	K1
fwd_rnd	C0/J	K1/C1	C0/J	K1/C1	R	T	T	R	T	K1	C0/J	C0/J	K1	T	K1/C1	K1/C1
restore_key	K1	C1	K1	C1	R	J	J	R	J	J	K1	K1	J	J	C1	C1
ksc4	K1/K2	C1	K1/K2	C1	R	T	T	R	J	J	K1/K2	K1/K2	J	J	C1	C1
backup_key	K2	C1	K2	C1	R	J	J	R	J	K2	K2	K2	K2	J	C1	C1
fwd_rnd	K2/C2	C1/J	K2/C2	C1/J	R	T	T	R	T	K2	K2/C2	K2/C2	K2	T	C1/J	C1/J

ksc4: generate next encryption round key from current
backup_key: copy entire 128-bit key into 2 64-bit registers
(n.b.: code uses 32-bit registers elsewhere to take advantage of superscalar archs)
fwd_rnd: performs one round of encryption
restore_key: copy 128-bit key to 4 32-bit regs (from 2 64-bit)

C#: ciphertext round #
K#: # round key
R: reserved
T: temporary usage
J: junk data

Figure 2: Register Usage of Loop-Amnesia (2 rounds of 10 shown)

From a security standpoint, the disk volume keys would frequently be transferred unencrypted over a bus from the TPM to the system CPU. An adversary able to tap this bus would be able to obtain the disk volume keys. From a performance standpoint, the master key would be decrypted by a relatively slower algorithm on a relatively slower processor, and we would in addition incur the latency of two transmissions over the TPM-CPU bus for every volume key decryption.[8] For these reasons, we chose not to utilize a TPM for our implementation.

5.2 Implementation Outline

The `aes_encrypt` and `aes_decrypt` functions take an AES context structure, a buffer containing the plaintext or ciphertext, and a buffer to which the encrypted ciphertext or decrypted plaintext must be stored. Each of these functions must use the master key to decrypt the volume key stored in the AES context structure, use this decrypted key to encrypt the plaintext buffer or decrypt the ciphertext buffer, and must finally write the fully encrypted ciphertext or fully decrypted plaintext to the output buffer. Programming these cryptographic routines in assembly language, on an architecture with 16 registers, and under the constraint that RAM not be used for working storage proved, predictably, to be a significant engineering challenge.

`aes_encrypt` and `aes_decrypt` work similarly as the encryption and decryption operations are nearly symmetric. There are 16 registers available for use on x86-64. Of these, `RSP` is the stack pointer and must always point to the stack, so it is not available for our use. We use `RBP` to point to the encryption or decryption function, depending on which operation we wish to perform. The 16 bytes of partially encrypted plaintext or partially decrypted ciphertext are moved from `EAX`, `ECX`, `R10D`, and `R11D` to `EBX`, `EDX`, `R14D`, and `R15D` during the performance of a single round of encryption or decryption. The routine performing a single round of encryption or decryption uses `R8`, `R13`, `RDI`, and `RSI` as temporary registers. The round key is stored in `R9` and `R12` while each round is performed. See Figure 2 for an illustration of Loop-Amnesia's register usage.

Thus, every general-purpose integer register available in the x86-64 instruction set is in use during the encryption and decryption subroutines. Since the 32-bit x86 architecture has only 8 integer registers available, adapting this technique to 32-bit x86 would likely require the use of the MMX or SSE registers. Adopting the technique to a RISC architecture with an abundance of general-purpose registers, however, would be straightforward.

`aes_set_key` is the routine to initialize an AES context structure with a given key. Our implementation generates the master key, if necessary, and initializes the AES context structure in RAM with the first and last round keys, first encrypting each with the master key.

6. VERIFICATION OF COLD-BOOT IMMUNITY

6.1 Justification

A system will be immune to a cold-boot attack if, when the system is running normally (i.e., not including directly after the input of a key to the system), no key data is ever stored to RAM. From the perspective of the x86-64 assembler programmer, key data could only be stored to RAM due to one of the following occurrences:

1. An explicit store, including a stack push instruction.
2. A taken interrupt causing registers with key data to be stored to the interrupt stack.

A review of the code in `aes_encrypt` and `aes_decrypt` easily shows that no register containing part of any master key, volume key, or round key is ever stored to RAM. Moreover, interrupts are disabled before the master key is read out of the MSRs and only enabled after registers containing key data have all been zeroed, so it is theoretically impossible for Loop-Amnesia to be vulnerable to the cold-boot attack given its structure.[9]

While perhaps not strictly necessary for immunity to the cold-boot attack, it is also not desirable that partially encrypted ciphertext (such as after one round of encryption) be stored to RAM as an attacker may be able to use cryptanalysis against such a degenerate version of AES to recover the volume key. Loop-Amnesia only stores fully encrypted ciphertext or fully decrypted plaintext to RAM, thwarting such an attack.

6.2 Correctness Testing

We performed correctness testing on an AMD Athlon64 X2 Dual Core Processor 3800+[10]. For convenience, we used the Linux /dev/mem device to inspect the physical RAM of this machine, rather than actually replicating the cold-boot attack ourselves. Using this methodology, we were able to extract the secret key from loop-AES. When using Loop-Amnesia, we found neither the master key nor volume key

[8]Performance problems due to bus latency and TPM processor speed would plague even a hypothetical TPM implementation supporting AES or another symmetric encryption algorithm.

[9]Non-maskable interrupts, or NMIs, cannot be disabled by software, and it is therefore theoretically possible for key data to leak to RAM if NMIs must be considered. We further discuss the problem of non-maskable interrupts in §8.3.

[10]using only one core, for the reasons mentioned in §5

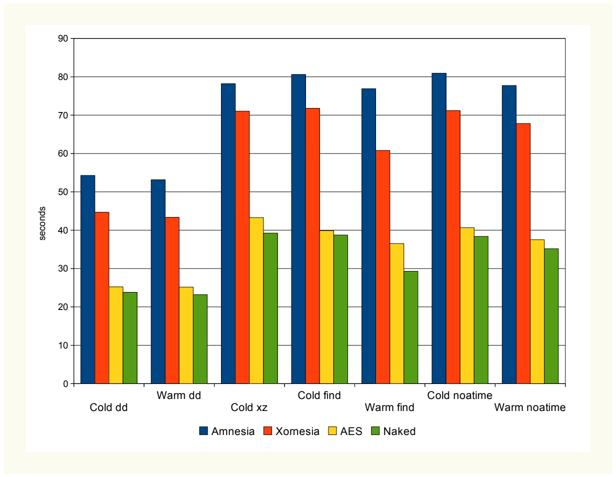


Figure 3: Loop-Amnesia Performance

present in RAM. We did, however, find data equivalent to the volume key encrypted with the master key present in RAM, as we expected.

7. PERFORMANCE

7.1 Benchmarking

We compare Loop-Amnesia against three other disk encryption methods. Our results are shown in Figure 3. "Xornesia" refers to a modified version of Loop-Amnesia which encrypts the disk volume keys in RAM by XORing them with the master key instead of performing full AES. Xornesia continues to use full AES when using the disk volume keys to do encryption and decryption of user data. We use Xornesia to isolate the overhead caused by repeated calculation of the key schedule, which is still present in Xornesia, from the overhead caused by the need to repeatedly decrypt the disk volume keys, which is not. "AES" refers to the loop-AES 128-bit AES implementation, with which we are fully compatible. We use this to measure the overhead of our Loop-Amnesia implementation relative to state-of-the-art disk encryption software using an optimized implementation of the same algorithm. "Naked" refers to a simple loopback mount with no encryption whatsoever. We use this as our baseline in order to eliminate from consideration the overhead of a loopback device.

The benchmarks are small, disk-intensive shell operations. dd writes a 900MB file consisting entirely of zeroes to disk. xz untars the Linux kernel from an xz-format archive. The "find" benchmark searches the Linux kernel source tree for instances of a particular word. "noatime" is the same as "find" but done on a filesystem mounted with an option to disable the recording of the time of last access. "Cold" benchmarks are done with the disk cache cleared; "warm" benchmarks are done after the disk cache has been primed by performing the same benchmark immediately before the test. We do not report numbers for warm xz as the CPU component of decompression made this test a poor measure of disk performance. We formatted the encrypted loopback device with the ext2 filesystem for all tests and used a single-core laptop with an Intel Celeron 540 at 1.8GHz with 1GB of RAM for benchmarking. The disk, a Hitachi HTS54258 (5400 RPM), experimentally performs reads at 725MB/s from the disk cache (on CPU) and at 44MB/s from the disk buffer (on disk microcontroller). Our results show that, on average, Loop-Amnesia introduces a slowdown of approximately 2.04x relative to Loop-AES and 2.23x relative to an unencrypted disk.

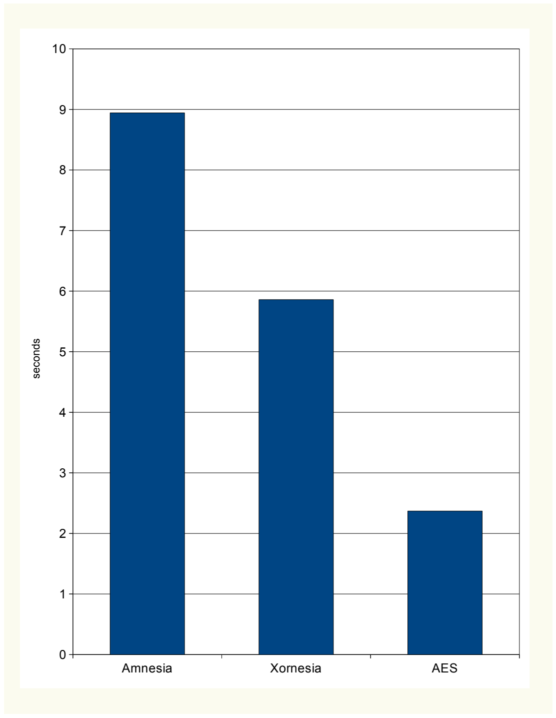


Figure 4: Amnesia, Xornesia, and AES CPU time

We also ran a simple unit test pitting Loop-Amnesia, Xornesia, and Loop-AES against each other, graphed in Figure 4. Since this is a CPU test, not a test of performance in practice, this provides a measure of the theoretical worst potential overhead Loop-Amnesia could cause, which would occur if disk accesses were free and performance of an encrypted filesystem was therefore bound entirely by CPU speed. The times given are for 10 million encryption and decryption operations. The theoretical worst-case slowdown of Loop-Amnesia relative to Loop-AES was found to be 3.77x.

7.2 Analysis

While we would have preferred Loop-Amnesia to have less of a performance impact, we believe that this overhead is acceptable given the unique benefit we provide. It is also worth noting that, while we designed these benchmarks to stress the disk subsystem, disk access speed does not play a major role in overall performance for many computing applications. The author has been using Loop-Amnesia for several months on both the laptop used for conducting the benchmarks and on another machine and has not noticed an appreciable decline in performance on either machine for interactive desktop use.[11]

[11]The machines did not previously use any form of disk encryption.

Our overhead comes from two sources. First, we must perform two cryptographic operations for each single cryptographic operation we are called on to perform by the loop-AES framework. Specifically, we must decrypt the device key with the master key, then use this decrypted key to perform the cryptographic operation originally requested (either encryption or decryption of a 16-byte block of data). Xornesia stores device keys XORed with the secret key rather than performing AES to encrypt the device keys, therefore cutting out the overhead of two cryptographic operations for every single act of encryption or decryption. Our second source of overhead is the necessity of generating round keys on-the-fly; loop-AES pregenerates these and keeps them within the AES context structure.

Though Xornesia has significantly lower overhead, we do not recommend the use of Xornesia instead of our original algorithm as doing so would weaken our security guarantee. An adversary able to choose the device key for an encrypted loopback device on the system would be able to derive the master key by performing the cold-boot attack and examining the encrypted device key.[12] From this, the attacker could discover the keys for encrypted loopback devices he did not configure. We felt that our method of defeating the cold boot attack should thwart even an attacker with user-level access to the machine.

Finally, we would like to note that, though our prototype does not make use of AES-NI, there is no technical barrier which would prevent AES-NI support from being added to Loop-Amnesia in a very straightforward manner. The only reason we did not make use of AES-NI is due to lack of access to a machine with AES-NI support for testing. Since AES-NI support has been shown to provide a more than a 10x speedup in other work [27], the performance cost of Loop-Amnesia after the addition of AES-NI support would be negligible.

8. LIMITATIONS

In this section, we discuss some limitations and potential vulnerabilities of both our approach and of the current implementation of Loop-Amnesia.

8.1 Architecture Dependence

Our approach is inherently architecture-dependent and limited to encryption systems with a kernel-mode implementation. An assembly-language implementation must be completed for every combination of CPU architecture and encryption algorithm needing support.

However, we nevertheless feel our approach is applicable to a wide variety of use cases. Encryption algorithms are small, self-contained pieces of code which only need be written once. Our implementation already supports a secure and widely used algorithm for the most common desktop and server CPU architecture. We expect that vendors will have the resources to adapt their existing encryption algorithm implementations – which, as in the case of loop-AES, may have already been implemented in assembly langauge for performance purposes – to use the Loop-Amnesia method for countering cold-boot attacks if there is even moderate institutional demand.

[12]It may also be possible to find the secret key by performing cryptanalysis on the first and last round keys in RAM, but we could negate this vulnerability by storing only the last round key in RAM and computing the first round key from the last whenever encryption is required. This would still be faster than performing full AES.

8.2 Functionality Limitations

As the CPU registers, including the MSRs, are cleared when a computer is suspended to RAM, we cannot support suspension to RAM. It would be possible for an implementation of our technique to copy the master key to RAM before allowing the computer to suspend, but this would be ill-advised: in such an implementation, the contents of the master key would be at risk of discovery by a cold-boot attack if the attacker gained access to the suspended computer.

8.3 Potential Effectiveness Issues

Espionage.

An attacker able to install a keystroke logger or otherwise tamper with the victim computer may be able to deduce the key through espionage. While we do not protect against a keystroke logger, the use of two-factor authentication, supported by loop-AES and Loop-Amnesia, could reduce its effectiveness, and a trusted path [38] execution framework could be used to prevent an attacker from tampering with unencrypted binaries used to mount the encrypted disk.

Key Information in Userspace.

According to the developer of loop-AES, the userspace portions of the cryptographic system of which Loop-Amnesia is a part will overwrite userspace key material with zeroes after transmitting it to the kernel [32]. However, since key material is transmitted through a UNIX pipe, it may still be available in the buffer unless the pipe is zeroed by the kernel after use; this is currently not done.

Cached Data.

Large amounts of decrypted data may be cached to RAM by the operating system, and our approach does not protect this data against a cold boot attack. However, it is possible for a user to manually clear the Linux disk cache by writing to a special file [4]. Periodically writing to this file from userspace, therefore, could mitigate the effectiveness of this attack at the expense of performance if the Linux kernel clears pages when they are freed (instead of when they are allocated). We have not checked whether the Linux kernel does in fact clear freed pages, but it would be simple to modify the operating system to do so.

JTAG.

Many processors implement a standardized debugging infrastructure called the Joint Test Action Group, or JTAG. By sending signals to a CPU over JTAG, a hardware developer is able to test the CPU's functioning. JTAG is commonly used in verifying that a particular CPU is not defective before releasing it for purchase. Because it is possible to use JTAG to dump the internal registers of a CPU, an attacker able to access the JTAG debug port may be able to read the Loop-Amnesia master key from the CPU's MSRs. Fortunately, it is rare for the JTAG debug port to be wired out for x86 processors [7]. In the rare case that a JTAG port is available on an x86 machine, we would recommend that a user concerned about this remove or destroy the JTAG port and/or blow the JTAG security fuse. Either of these actions would disable an attacker's ability to access JTAG [24].

Non-Maskable Interrupts.

We take care to disable interrupts before reading the master key into general-purpose registers and to reenable them only after the key has once again been erased from all general-purpose registers. However, some interrupts, called non-maskable interrupts (NMIs), cannot be disabled. These interrupts are usually caused only by hardware faults. Since the general-purpose registers are stored to RAM when an interrupt is taken, an attacker able to introduce a hardware fault during the brief time periods when key material is in the general-purpose registers would be able to read the master key. We consider such an attack unlikely to prove practical, primarily due to its complexity and dependence on extreme luck in timing. However, if this attack does prove to be a concern, modifying the operating system's interrupt handler to scrub the general-purpose registers from RAM after receiving a non-maskable interrupt would be sufficient to protect against it. This would have no deleterious side effects as the hardware will have faulted, so the CPU will never resume normal execution.[13]

9. RELATED WORK

9.1 Lest We Remember: Cold-Boot Attacks on Encryption Keys

Halderman et al. discussed some forms of mitigation in [23], including deleting keys from memory when an encrypted drive is unmounted[14], obfuscation techniques, and hardware modifications such as intrusion-detection sensors and epoxy-encased RAM. Halderman et al. admit that they do not present a full solution applicable to general-purpose hardware.

While special-purpose hardware modifications may be effective, such hardware adds cost and may not be available to many users of disk encryption; a solution for commodity hardware is required. As the cold-boot attacker is given a copy of all RAM, including the program text used to perform encryption and decryption, we doubt that obfuscation would prove effective.

9.2 TRESOR and AESSE

A paper at Eurosec 2010 [26] discussed a potential solution to the cold boot attack, in which a single encryption key was stored in the SSE registers of the CPU and SSE register access was disabled for user-level code. In a USENIX Security 2011 paper [27], the same authors extended their prototype, called TRESOR, with the ability to make use of AES-NI when available.[15] A fundamental limitation of TRESOR relative to Loop-Amnesia is its inability to support multiple disk encryption keys, since only one encryption key schedule may be stored inside the SSE registers. In addition to common use cases such as encrypted data and swap partitions, the limitation of a single protected disk encryption key forecloses the use of multiple keys as a protection against watermarking attacks, such as is done in loop-AES [31].

Another limitation of TRESOR relative to Loop-Amnesia is TRESOR's need to disable SSE for use as key storage. The authors note that many applications which use SSE when available do not correctly check for its availability and crash when run on a TRESOR-modified kernel. Moreover, SSE is necessary for many common applications, including multimedia applications and, as TRESOR's authors point out, current versions of OpenGL. Finally, many applications that do not require SSE can nevertheless make use of it if it is available; the loss of its availability will cause these applications to suffer a performance degradation. The compatibility and performance problems with TRESOR due to its use of the SSE registers for key storage are serious issues and are remedied by Loop-Amnesia's use of MSRs.

9.3 Braving the Cold Black Hat Talk

A talk [25] at Black Hat in 2008 discussed various methods of mitigating the effects of the cold boot attack. Most of these mitigation strategies are discussed elsewhere; however, one contribution of this talk is a suggestion that motherboard temperature sensors be used to detect attempts to cool RAM and take protective measures, such as scrubbing the keys.

This talk also proposed a potential solution to the cold boot attack. The researchers suggested that the key could be stored in RAM only as the product of the hash of a large block of bits. The hope is that at least one of these bits will flip during the performance of the cold boot attack, preventing its success. This strategy, if implemented, would likely suffer from severe performance problems as a large hash would need to be calculated every time an encryption key needed to be accessed. The talk also discussed "caching" the encryption key inside the MMX registers, but it was unclear from the talk how such a caching system would operate.

9.4 Frozen Cache

Jürgen Pabel has posted a website [29], dormant since early 2009, detailing his plans to provide a software-based solution to the cold-boot attack. His approach is to memory-map the L1 cache of the CPU and use this space to store the AES key schedule. Because this approach would prevent the CPU cache from serving its normal role, every memory access on the machine would result in a cache miss. Disabling the CPU cache in this manner results in a slowdown of perhaps 200x [1] felt by all software, not just software accessing files on the encrypted disk. Because our solution avoids the negative system performance side effects of Pabel's design, we believe it to be more practical.

9.5 Linux-Crypto Mailing List Brainstorming

Shortly after Halderman et al. published their attack, a mailing list discussion on Linux-Crypto discussed possible mitigation strategies. The general approach of keeping key information in CPU registers was brought up [40], but the ideas given were too vague to suggest how this might specifically be accomplished and do not appear to have been pursued further.

9.6 Leakage-Resistant Algorithms

There has been considerable work [6] [17] [28] [37] in designing cryptosystems resilient to partial key leakage due to

[13]Our prototype implementation does not modify the OS interrupt handlers.

[14]this is already done in loop-AES according to [32]

[15]As previously noted, there is no technical barrier which would prevent Loop-Amnesia from making use of AES-NI, Via PadLock, or a similar instruction set extension as well, though the prototype does not currently support these ISA extensions.

side channel attacks. Most of this work has focused on the design of new ciphers with properties mitigating the impact of partial key leakage.

Unfortunately, we do not believe that protecting against partial key leakage is a sufficient defense against the cold boot attack. According to Halderman et al., it is possible to perform the cold-boot attack in such a way that over 99.9% of memory remains uncorrupted an entire minute after power is cut. Any countermeasure to the cold-boot attack must account for its potential to fully leak any encryption keys stored to RAM.

9.7 TCG Platform Reset Attack Mitigation Specification

The Trusted Computing Group has published a standard [21] which purports to mitigate the vulnerability of compliant systems to the cold-boot attack. This specification states that a compliant BIOS must zero out all RAM before giving control to the operating system. While this prevents the attack from being performed using only the victim's computer[16], the attacker can still easily perform the attack by moving the RAM from the victim's machine to a machine under his own control, then booting using a BIOS not following the TCG specification. Thus, the TCG specification cannot be considered a sufficient countermeasure.

9.8 Forenscope: A Framework for Live Forensics

The RAM of a computer may contain sensitive material other than the encryption keys to the hard disk. The Forenscope rootkit [10] takes advantage of the cold-boot attack to gain access to active network sessions as well; the session keys' presence in memory could allow an attacker to masquerade as the victim to any website, SSH server, or other remote system to which the user was connected at the time of the attack.

Loop-Amnesia will protect against a Forenscope-using attacker's gaining access to the encrypted disk: the attack tool uses the exact same strategy as Halderman et al. to attempt recovery of the key. Unfortunately, SSH and SSL session keys will likely remain in RAM, so an attacker with Forenscope could still conceivably keep the victim's network connections alive, sniff the session keys, and masquerade as the victim to connected machines. See §10 for a discussion on how Loop-Amnesia may be extended to assist in preventing Forenscope attacks.

10. FUTURE WORK

A ripe area for future research is the applicability of our approach to algorithms outside of the AES cipher family. Some algorithms, such as Blowfish [34], use key-dependent S-boxes; proving whether these S-boxes can be safely stored to RAM would require careful analysis. We believe that our approach should work well for all algorithms without key-dependent S-boxes and with key schedules that are computationally inexpensive to compute, but its effectiveness outside this class of ciphers remains to be analyzed.

The ability of Loop-Amnesia to assist in neutralizing Forenscope's other attack capabilities also merits examination. For instance, an operating system attempting to harden itself against Forenscope could use Loop-Amnesia to encrypt various pieces of data inside the kernel TCP stack. As the master key will have been erased by the reboot preceding Forenscope's installation, Forenscope will have no way of recovering the network connections. By the time the attacker has had time to download and analyze the SSH/SSL session keys from RAM, any active TCP sessions will likely have expired.

Finally, our work exposes a limitation in current system programming languages: the inability to insist to a compiler that particular values never be spilled to RAM. While we recognize that our needs are uncommon and do not by themselves merit the redesign of system programming languages, we speculate that programming language designers may one day wish to allow users more control over the register allocation process for performance reasons. We would encourage the designers of such languages or language extensions to include functionality allowing the user to express the needs we faced when implementing Loop-Amnesia. User control over the register allocation process may provide useful benefits for both security and performance.

[16]A BIOS password would also necessitate the use of a separate machine.

11. CONCLUSION

In this paper, we present one of the first practical solutions to the cold-boot attack applicable to general-purpose hardware. For a performance cost likely to be very moderate under most workloads, our solution provides protection for general-purpose hardware against a significant practical attack affecting all previous state-of-the-art disk encryption systems. We present a design strategy applicable to all operating system-based disk encryption systems and a usable open-source implementation which validates our design. After the publication of this paper, we intend to work with the Linux kernel community to integrate our approach, and possibly code, into the standard Linux kernel distribution.

12. ACKNOWLEDGEMENTS

We thank Andrew Lenharth of the University of Texas at Austin for his invaluable inspiration and advice in the early stages of this work. We also thank Jari Ruusu for providing loop-AES to the free and open source software community: being able to use such well-designed software as the base for our implementation significantly aided us in evaluating the concepts behind Loop-Amnesia.

13. REFERENCES

[1] Cachegrind: a cache-miss profiler. `http://wwwcdf.pd.infn.it/valgrind/cg_main.html`.

[2] Truecrypt: Free open-source on-the-fly encryption. `http://www.truecrypt.org/`.

[3] Hard drive secrets sold cheaply. `http://news.bbc.co.uk/2/hi/technology/3788395.stm`, June 2004.

[4] drop_caches. `http://www.linuxinsight.com/proc_sys_vm_drop_caches.html`, May 2006.

[5] Privacy at risk after burglary at doctor's office. `http://www.cbc.ca/health/story/2011/01/21/nb-privacy-warning.html`, January 2011.

[6] Adi Akavia, Shafi Goldwasser, and Vinod Vaikuntanathan. Simultaneous hardcore bits and

cryptography against memory attacks. In *Theory of Cryptography Conference*, pages 474–495, 2009.

[7] Mike Anderson. Using a JTAG in linux driver debugging. In *CE Embedded Linux Conference*, 2008. http://elinux.org/images/4/4e/CELF_JTAG_Anderson.ppt.

[8] Alex Biryukov and Dmitry Khovratovich. Related-key cryptanalysis of the full AES-192 and AES-256. Cryptology ePrint Archive, Report 2009/317, 2009. http://eprint.iacr.org/.

[9] Bob Brown. How to roll out full disk encryption on your pcs and laptops. http://www.networkworld.com/news/2010/081610-encryption.html, August 2010.

[10] E. Chan, S. Venkataraman, F. David, A. Chaugule, and R. Campbell. Forenscope: A framework for live forensics. In *Annual Computer Security Applications Conference*, November 2010.

[11] Intel Corporation. IA-32 architectural MSRs. *Intel 64 and IA-32 Architectures Software Developer's Manual*, 3B:681–722, January 2011. http://www.intel.com/Assets/PDF/manual/253669.pdf.

[12] Microsoft Corporation. Bitlocker drive encryption technical overview. *Microsoft Technet*, 2010. http://technet.microsoft.com/en-us/library/cc732774\%28WS.10\%29.aspx.

[13] John Criswell, Andrew Lenharth, Dinakar Dhurjati, and Vikram Adve. Secure virtual architecture: a safe execution environment for commodity operating systems. In *Proceedings of Twenty-First ACM SIGOPS Symposium on Operating Systems Principles*, SOSP '07, pages 351–366, New York, NY, USA, 2007. ACM.

[14] John Curran. Encrypted laptop poses 5th amendment dilemma. *USA Today*, February 2008. http://www.usatoday.com/tech/news/techpolicy/2008-02-07-encrypted-laptop-child-porn_N.htm.

[15] Joan Daemen and Vincent Rijmen. *The Design of Rijndael*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2002.

[16] Advanced Micro Devices. MSRs of the AMD64 architecture. *AMD64 Architecture Programmer's Manual*, 2:469–472, June 2010. http://support.amd.com/us/Processor_TechDocs/24593.pdf.

[17] Stefan Dziembowski and Krzysztof Pietrzak. Leakage-resilient cryptography. In *FOCS*, pages 293–302, 2008.

[18] Jake Edge. Holes in the linux random number generator? *Linux Weekly News*, 2006. http://lwn.net/Articles/184925/.

[19] David W. Foley. http://doj.nh.gov/consumer/pdf/wackenhut.pdf, December 2010.

[20] Henri Gilbert and Thomas Peyrin. Super-sbox cryptanalysis: Improved attacks for aes-like permutations. Cryptology ePrint Archive, Report 2009/531, 2009. http://eprint.iacr.org/.

[21] Trusted Computing Group. TCG platform reset attack mitigation specification. http://www.trustedcomputinggroup.org/resources/pc_client_work_group_platform_reset_attack_mitigation_specification_version_10/, 2008.

[22] Zvi Gutterman, Tzachy Reinman, and Benny Pinkas. Analysis of the linux random number generator. In *IEEE Symposium on Security and Privacy*, 2006.

[23] J. Alex Halderman, Seth D. Schoen, Nadia Heninger, William Clarkson, William Paul, Joseph A. Calandrino, Ariel J. Feldman, Jacob Appelbaum, and Edward W. Felten. Lest we remember: Cold boot attacks on encryption keys. In Paul C. van Oorschot, editor, *USENIX Security Symposium*, pages 45–60. USENIX Association, 2008.

[24] Zack Albus Markus Koesler, Franz Graf. Programming a flash-based msp430 using a JTAG interface. http://www.softbaugh.com/downloads/slaa149.pdf, December 2002.

[25] Patrick McGregor, Tim Hollebeek, Alex Volynkin, and Matthew White. Braving the cold: New methods for preventing cold boot attacks on encryption keys, 2008.

[26] Tilo Müller, Andreas Dewald, and Felix C. Freiling. Aesse: a cold-boot resistant implementation of aes. In *Proceedings of the Third European Workshop on System Security*, EUROSEC '10, pages 42–47, New York, NY, USA, 2010. ACM.

[27] Tilo Müller, Andreas Dewald, and Felix C. Freiling. Tresor: Tresor runs encryption securely outside ram. In *USENIX Security Symposium 2011*, 2011.

[28] Moni Naor and Gil Segev. Public-key cryptosystems resilient to key leakage. In *Proceedings of the 29th Annual International Cryptology Conference on Advances in Cryptology*, pages 18–35, Berlin, Heidelberg, 2009. Springer-Verlag.

[29] Jürgen Pabel. http://frozencache.blogspot.com, 2009.

[30] OpenSolaris Project. ZFS on-disk encryption support. http://hub.opensolaris.org/bin/view/Project+zfs-crypto/WebHome.

[31] Jari Ruusu. http://loop-aes.sourceforge.net/.

[32] Jari Ruusu. http://mail.nl.linux.org/linux-crypto/2008-06/msg00002.html, June 2008.

[33] Christophe Sauot. dm-crypt: A device-mapper crypto target. http://www.saout.de/misc/dm-crypt/.

[34] Bruce Schneier. Description of a new variable-length key, 64-bit block cipher (blowfish). In *Fast Software Encryption, Cambridge Security Workshop*, pages 191–204, London, UK, 1994. Springer-Verlag.

[35] Freescale Semiconductor. Performance monitor counter registers. *MPC750 RISC Processor Family User's Manual*, pages 378–382, December 2001. http://www.freescale.com/files/32bit/doc/ref_manual/MPC750UM.pdf.

[36] Arvind Seshadri, Mark Luk, Ning Qu, and Adrian Perrig. SecVisor: a tiny hypervisor to provide lifetime kernel code integrity for commodity OSes. *SIGOPS Oper. Syst. Rev.*, 41:335–350, October 2007.

[37] Francois-Xavier Standaert, Olivier Pereira, Yu Yu, Jean-Jacques Quisquater, Moti Yung, and Elisabeth Oswald. Leakage resilient cryptography in practice. In David Basin, Ueli Maurer, Ahmad-Reza Sadeghi, and David Naccache, editors, *Towards Hardware-Intrinsic Security*, Information Security and Cryptography, pages 99–134. Springer Berlin Heidelberg, 2010.

[38] Allan Tomlinson. Introduction to the TPM.

`http://courses.cs.vt.edu/cs5204/fall10-kafura-BB/Papers/TPM/Intro-TPM-2.pdf`.

[39] Zhi Wang and Xuxian Jiang. Hypersafe: A lightweight approach to provide lifetime hypervisor control-flow integrity. In *IEEE Symposium on Security and Privacy*, pages 380–395, 2010.

[40] Richard Zidlicky. `http://www.spinics.net/lists/crypto/msg04668.html`, 2008.

Private Search in the Real World

Vasilis Pappas, Mariana Raykova, Binh Vo, Steven M. Bellovin, Tal Malkin
Department of Computer Science
Columbia University, New York, NY, USA
{vpappas, mariana, binh, smb, tal}@cs.columbia.edu

ABSTRACT

Encrypted search — performing queries on protected data — has been explored in the past; however, its inherent inefficiency has raised questions of practicality. Here, we focus on improving the performance and extending its functionality enough to make it practical. We do this by optimizing the system, and by stepping back from the goal of achieving maximal privacy guarantees in an encrypted search scenario and consider efficiency and functionality as priorities.

We design and analyze the privacy implications of two practical extensions applicable to any keyword-based private search system. We evaluate their efficiency by building them on top of a private search system, called SADS. Additionally, we improve SADS' performance, privacy guaranties and functionality. The extended SADS system offers improved efficiency parameters that meet practical usability requirements in a relaxed adversarial model. We present the experimental results and evaluate the performance of the system. We also demonstrate analytically that our scheme can meet the basic needs of a major hospital complex's admissions records. Overall, we achieve performance comparable to a simply configured MySQL database system.

1. INTRODUCTION

Encrypted search — querying of protected data — has come into the foreground with growing concerns about security and privacy. There are many variants of the problem that protect different things: the searchable data, queries, participant identities, etc. Existing schemes also differ in their expected operational environment. The majority of encrypted search mechanisms concern data outsourcing [4–6,8,13,33,34] and to a lesser degree data sharing [9,17,30]. Data outsourcing concerns the case where one party wants to store its encrypted data on an untrusted server and be able to search it later. Data sharing involves one party who provides limited search access to its database to another. These two settings require different privacy guarantees of an encrypted search system; data outsourcing is not concerned with protecting the data from the querier, since he is the owner. Furthermore, specific implementations may return different things (e.g., number of matches, document identifiers, related content, etc.) or may differ in number of participants, trust assumptions, anonymity requirements, revocation of search capability and other areas. All of these factors affect performance. Choosing a different definition of "sufficient" privacy can greatly affect inherent cost. Making the right choice, in accordance with the actual, rather than theoretical, threat model can lead to a very functional system, rather than one that is theoretically perfect but unusably costly in practice.

In this paper we step back from absolute privacy guarantees in favor of efficiency and real-world requirements. These requirements include not just what may leak, but to whom; depending on the particular practical setting there may be parties who are at least partially trusted. Our goal is to describe and build systems that meet the privacy guarantees matching the actual goals for a given scenario, so that we may improve efficiency. Towards this end, we present a set of generic extensions, applicable to any keyword-based private search system. We discuss the importance of each of these, the challenges for their secure implementation and analyze their privacy implications in terms of leakage. To evaluate their efficiency, we developed them on top of SADS [30], an efficient private search system that uses Bloom filters. In addition, we describe and implement a number of new features in SADS that improve its performance, privacy guarantees and functionality. Finally, we describe and analyze the performance of the extended SADS system in a real-world scenario, using health records.

Our implementation and the obtained empirical results are an important contribution of this paper from the point of view of evaluating the real usability of the proposed system for practical purposes. Although theoretical analysis asserts that a Bloom filter-based search should be efficient, it is unwise to rely solely on theory. If nothing else, complexity analysis says nothing about constant factors, and says nothing about unexpected bottlenecks. It matters little if an algorithm has n^3 exponentiations if n is reasonably small and the entire system is in fact I/O-bound rather than CPU-bound [24]. Similarly, Kernighan and Pike noted that "measurement is a crucial component of performance improvement since reasoning and intuition are fallible guides and must be supplemented with tools" [23]. Our work shows that — asymptotic behavior aside — our scheme is practical across a wide range of input sizes. Equally important, it shows the cost of different kinds of privacy. Neither conclusion is amenable to a purely theoretical

This material is based on research sponsored by Air Force under agreement number FA8750-09-1-0075. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of Air Force or the U.S. Government.

analysis.

The contributions of this work are:

- We present two practical extensions, namely Document Retrieval and Range Queries, that can be used on top of any keyword-based private search system.
- We improve an existing private search system (SADS) to provide better privacy, support data updates and become more robust.
- We implement all of the above and provide extensive evaluation results and a case study. Code, datasets and data inputs are available online at `http://nsl.cs.columbia.edu/projects/sads/` for similar systems to compare with.

2. BACKGROUND

2.1 Secure Anonymous Database Search

The secure anonymous database search (SADS) scheme [30] provides the following search capability: it allows a search client (C) with a keyword to identify the documents of a database owner/server (S) containing the keyword without learning anything more or revealing his query. For this purpose the architecture of the system involves two semi-trusted parties: index server (IS) and query router (QR), which facilitate the search. In summary the scheme works as follows: the database owner computes search structures for his database — a Bloom filter (BF) per document built from the encryptions of all words of the document. Each authorized client receives keys that he uses to submit queries and decrypt the results; the QR receives corresponding transformation keys for the queries of that client. To submit a query, C computes an encryption of his query and sends it to QR. QR verifies that the client is authorized, re-encrypts the query with the corresponding transformation key, computes and sends the BF indices obtained from the encryption to IS. IS performs search across the BFs it stores, encrypts the identifiers of the matching documents and sends them to the QR; QR transforms the encryptions and delivers them to the client, who decrypts them to obtain his search results (see Figure 1).

The original implementation of SADS also includes a couple of optimizations/features enabled by the use of BFs. First, storing the BFs in transposed order – called *slicing optimization* – minimizes the number of bits that need to be read during search. That is because only bit slices corresponding to specific indices are read during a query and not all the BFs. This approach has two main benefits. First, it has better cache behavior because it fetches each slice once and uses it for all the result vectors; second, in some cases it avoids reading several slice portions if the corresponding bits of all the result vectors have been zeroed out. In addition, SADS also supports *boolean queries*. One naive way to do this is to search for each term separately and union or intersect the results. However, BFs can more efficiently handle ANDs by combining indices into a superset, and ORs are handled in parallel by the slicing optimization.

2.2 Security Definitions and Relaxed Privacy Settings

The strongest security definition for a generic encrypted search scheme in the setting of data sharing guarantees that the querier receives only the matching results, while none of the other parties in the protocol learns anything. If we formalize this intuition by applying the standard simulation security notion of Canetti [7], what

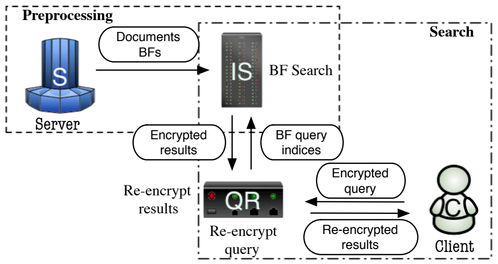


Figure 1: SADS overview.

the definition captures is that a protocol is secure if the views of the participants in the real execution (namely their inputs, random inputs, outputs, and messages they receive) are indistinguishable from their views in an ideal execution where all parties send their inputs to a trusted party who computes the results and sends them back to the receivers.

Satisfying this level of privacy inevitably comes at efficiency cost. In many scenarios, weaker privacy guarantees may be sufficient. In particular, to achieve better performance, it is often acceptable to leak some controlled amount of information. The next definition gives the general security notion for encrypted search with certain privacy leakage.

DEFINITION 1. *A protocol π is a secure encrypted search protocol with privacy leakage $\mathcal{L}$, if for every real world adversary, there exists a simulator such that the view of the adversary in the real world execution, where he interacts with the honest parties, is indistinguishable from his view in an ideal world execution where he interacts with a simulator that takes as input $\mathcal{L}$ (and simulates the honest parties).*

SADS has the following privacy leakage $\mathcal{L}$ with respect to the parties that perform the search (i.e., the IS and the QR if they collude):

- *False Positive Database Leak*: a fraction of records that do not match the search criterion
- *Search Pattern*: the equality pattern of the submitted queries
- *Results' Pattern*: the equality pattern among the results
- *Similarity Database Leak*: search structures leaking information about similarity of data records.

In Section 5.1 we present a modification to the scheme that removes the last type of leakage that comes just from the search structures on their own.

The above security guarantees apply to the search functionality for exact match queries. The SADS search scheme further supports Boolean queries, which provide privacy guarantees for the non-matching part of the database, i.e., the querier does not learn anything about the non-matching records. With respect to the query privacy from the search party the Boolean queries reveal the matching pattern over the different terms in the search query in addition to the results' pattern. In Section 4 we introduce a scheme that realizes range query functionality that is based on OR queries and inherits the query leakage from the Boolean queries. The leakage from the OR queries in the context of the range queries means that given a range query the index server will be able to obtain identifiers for each logarithmic-sized sub-range that two records have terms co-occuring in, starting from the unique value and ranging up to the sub-range equal to half the size of the full range. It does not, however, learn what ranges these are, or, their size. The identifiers

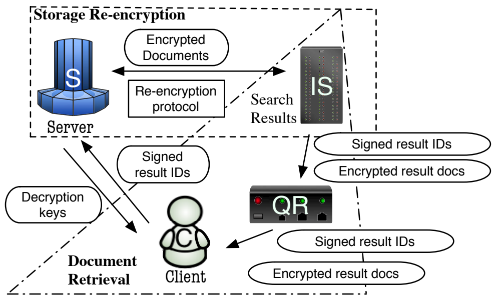


Figure 2: SADS with Document Retrieval.

can only be useful to determine patterns across multiple queries.

3. DOCUMENT RETRIEVAL

There exist many systems for searching databases to privately identify items of interest. An extension of obvious use is a system to then retrieve those items privately. One way to do this is with private information retrieval techniques, however these are very expensive, and can be even more expensive when fetching large numbers of records, or records of individually great size. We present a system that is much more efficient, at the cost of requiring a trusted third party, and can be modularly implemented to extend any private search system that returns handles representing matches.

Systems both with and without document retrieval have practical use. For example, a user may simply wish to establish that a server does have documents of interest to him, or may wish to determine how many are of interest, or learn about certain qualities concerning the data held there (subject to the search permissions granted by the server). Furthermore, even in systems that include document retrieval, separating this functionality from query is worthwhile. For example, the server may be running a paid service, and allow the user to operate in an initial stage wherein he determines what he wants, and a bargaining stage wherein they negotiate pricing, before purchasing the actual content.

Document retrieval poses its own challenge, especially when the data is not owned by the party retrieving it. In this scenario, returning additional data is a privacy leak for the data owner; at the same time, revealing the matching documents to the owner is a privacy leak for the retriever. Thus, the strongest security we would want to aim for would require us to touch the contents of the entire database [9]. This is a prohibitively expensive cost for applications that aim to work in "real time" over a large data set. One way to avoid this cost is to relax our security definition and allow leakage of the retrieval pattern (i.e. whether separate retrieval attempts touched the same documents). In the case of data outsourcing, this amount of privacy leakage easily suffices, since the untrusted server just searches for and returns the encrypted files that he stores to the owner who has the corresponding decryption keys [4, 8, 13]. This approach, however, is not applicable to the case of data sharing, where leaking the matching documents to the owner reveals more than the result pattern: he also knows the content of the documents, from which he can infer information about the query.

This problem is similar to that addressed by private information retrieval protocols (PIR) [10, 16, 29], wherein a server holds a set of items from which a user wishes to retrieve one without revealing which item he is requesting. It differs slightly in that we wish to retrieve multiple items (corresponding to the search results). It also differs in that we require that the selected set be certified and that the user does not learn content of documents outside of it. There are PIR schemes that address this [16], but at additional cost. Thus, our problem could be addressed by simply running an appropriate PIR scheme once for each document result. However, PIR is already quite expensive for a single document, and running them multiply would only aggravate this.

We address this by constructing a document retrieval scheme that can be used on top of any other scheme that returns document IDs. Our scheme maintains efficiency by introducing an intermediary party who stores the encrypted files of the database and provides the matching ones to the querying party. This party is given limited trust to perform the search, but he should not be able to decrypt the stored files. In this case we need to provide the querier with the decryption keys for the result documents; these are known to the data owner, who must be able to provide the correct keys obliviously without learning the search results. In Figure 3 we present a protocol that realizes the document retrieval functionality between a data owner (S) and a client (C) with the help of an intermediary party (P). For the purposed of this protocol we assume that there is a search functionality $EncSearch$ that returns the IDs of the documents matching a query from the client. For a query Q we denote $EncSearch(Q)$ the returned set of document IDs. The database of the server that is used for the protocol consists of documents $D_1, \dots, D_n$. Our protocol also uses 1-out-of-n oblivious transfer (OT) functionality that allows two parties, one of which has input an array and the other has input an index in the array, to execute a protocol such that the latter party learns the array element at the position of his index and the former learns nothing. There are many existing instantiations of OT protocols, we use the protocol of [15], which allows best efficiency. The last tool for our constructions is an encryption scheme with the following property (defined in more detail in [30], which also gives an instantiation for such a scheme):

DEFINITION 2 (ENCRYPTION GROUP PROPERTY). *Let $\Pi = (GEN, ENC, DEC)$ be a private key encryption scheme. We say that Π has a group property if $ENC_{k1}(ENC_{k2}(m)) = ENC_{k1 \cdot k2}(m)$ holds for any keys k_1, k_2 and any message m.*

Intuitively, the security of this protocol is based on the secrecy of the permutation π, known only to P. Because it is not known to S, S cannot correlate the keys k'_{π_i} that are requested by C with the original indices of the matching documents. He learns only the search pattern of the querying party. We can take two approaches to mitigate this leakage. The querying party may aggregate requests for decryption keys to the server for the search results of several queries. Another solution is to extend the scheme to include additional keys pertaining to no real documents, which P can add to the sets of requested keys so that S cannot tell how many of the keys he returns correspond to query results. Step 2 of the re-encryption can be implemented using protocols for oblivious transfer [1, 11, 27].

4. RANGE QUERIES

We now present an extension that enables multi-dimensional range queries using any underlying private search system that, preferably, supports boolean queries in conjunctive normal form over exact string matches. We first describe our system as a general construction then discuss how some of the costs interact with the efficiency tradeoffs inherent in the SADS system due to the use of BFs.

4.1 General construction

This generic extension introduces the following additional costs

Storage Reencryption (***preprocessing*** *phase*)

Inputs:
$S:\ D_1, \ldots, D_n$, keys $k_1, \ldots, k_n$ and $k'_1, \ldots, k'_n$;
$P:$ permutation π of length n ;
$S, P:\ (\overline{GEN}, \overline{ENC}, \overline{DEC})$ satisfying Definition 2
Outputs:
$S: \perp;\ \ P:\ \overline{ENC}_{k'_{\pi(i)}}(D_i)$ for $1 \leq i \leq n$
Protocol:

1. S sends to P $c_i = \overline{ENC}_{k_i}(D_i)$ for $1 \leq i \leq n$.
2. For each $1 \leq i \leq n$ S and P execute 1-out-of-n OT protocol that allows P to obtain $k''_i = k_i^{-1} \cdot k'_{\pi(i)}$.
3. For each $1 \leq i \leq n$ P computes $\overline{ENC}_{k''_i}(c_i) = \overline{ENC}_{k_i^{-1} \cdot k'_{\pi(i)}}(\overline{ENC}_{k_i}(D_i)) = \overline{ENC}_{k'_{\pi(i)}}(D_i)$.

Document Retrieval

Inputs:
S : keys $k'_1, \ldots k'_n$;
P : permutation π of len n, $\overline{ENC}_{k'_{\pi(i)}}(D_i)$, $1 \leq i \leq n$;
C : query Q;
S, P, C : search scheme *EncSearch* that returns IDs of matched documents to P, C.
Outputs:
S : cardinality of the output set $EncSearch(Q)$;
P : IDs of docs matching query Q from *EncSearch*;
C : the content of the docs matching Q from *EncSearch*.
Protocol:

1. S, P, C run *EncSearch* for query Q. Let $i_1, \ldots, i_L$ be the IDs of the matching documents.
2. P sends $Sign(\pi(i_1), \ldots, \pi(i_L))$ to C together with the encrypted documents $\overline{ENC}_{k'_{\pi(i_1)}}(D_{i_1})$, $\ldots, \overline{ENC}_{k'_{\pi(i_L)}}(D_{i_L})$.
3. C sends $Sign(\pi(i_1), \ldots, \pi(i_L))$ to S.
4. S verifies $Sign(\pi(i_1), \ldots, \pi(i_L))$ and returns $k'_{\pi(i_1)}, \ldots, k'_{\pi(i_L)}$.
5. C decrypts $\overline{ENC}_{k'_{\pi(i_1)}}(D_{i_1}), \ldots, \overline{ENC}_{k'_{\pi(i_L)}}(D_{i_L})$ to obtain the result documents.

Figure 3: Protocol for Document Retrieval

over its underlying search system:

• For each inserted point, we will need to insert into the underlying search system $d \lg r$ terms, where d is the number of dimensions we are supporting and r is the size of the range of values supported per dimension.

• For each query, we will need to issue a boolean query using up to $2d\lg(\frac{q}{2})$ query terms to the underlying search system, where q is the size of the range being queried.

• The system presents repeatable unique identifiers for logarithmically cut sub-regions across all documents in a single query, and across multiple queries. If the underlying private search system does not guarantee full privacy of its queries, this can increase the information leakage over what would normally be incurred.

Our basic approach is to represent each ranged dimension as a binary value. Then, for each one, we create a strata for each digit of the value, and for each strata, divide the range into binary pieces according to the order of the digit, and assign each piece of each strata of each dimension a globally unique identifier. To insert a term to the search index, we insert the identifier of every piece that contains it (thus one term is inserted per dimension per strata, with a number of strata logarithmic in the size of the ranges). An example of inserting the value "11" using 4-bit numbers is hown in Figure 4 (top).

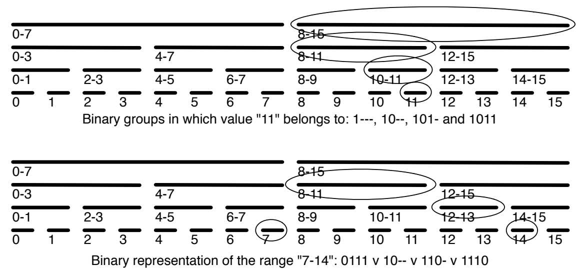

0-7
8-15
0-3
4-7
8-11
12-15
0-1
2-3
4-5
6-7
8-9
10-11
12-13
14-15
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
Binary groups in which value "11" belongs to: 1---, 10--, 101- and 1011
Binary representation of the range "7-14": 0111 v 10-- v 110- v 1110

Figure 4: Terms used for inserting the value "11" (top). Boolean query for range "7-14" (bottom).

To issue a query, we create a boolean OR query. For each dimension, we start at the strata with the largest and least numerous pieces, and add to the query the largest piece that fits entirely within the query range. We iterate to lower strata, adding pieces that fit entirely into the range without covering values that are already covered by existing pieces in the query, and continue, if necessary, to the lowest strata which contains every individual value in the full range. We then create an AND query across all of the dimensions, resulting in a query in conjunctive normal form. An example query on one dimension is shown in Figure 4 (bottom).

Since every single piece of every strata that contains the representative value has been added to the index, this query will return true if and only if the range query contains it. The worst case query, is for the query range to straddle the midway point of the full range. This results in taking $2\lg(\frac{q}{2})$ query terms per dimension.

THEOREM 1. *A contiguous range query on a single dimension cannot require more than* $2\lg(\frac{q}{2})$ *disjunctive terms.*

PROOF. We begin with an initial lemma: a contiguous query cannot require more than two terms in a single strata, one in its lower half and one in its upper half. Let us assume to the contrary that it did require two terms within a single bisection of its range. Then, starting from the uppermost term, the range contains a subrange equal to at least four times the size of the elements of the strata (two in each bisection). Since the strata above uses elements of twice the size, and there is at least one term within the range that is not one of the endpoints, that term is a subset of a range from the upper strata that is contained entirely within this subrange. Thus, the term representing that range could have been chosen instead to replace two or more terms representing smaller ranges, a contradiction.

Given that each strata uses ranges twice the size of the strata beneath it, it is trivial to show via summing that a query cannot require terms from more than $\lg(q) - 1$ strata. In conjunction with our lemma, we thus show that a contiguous range query cannot require more than $2(\lg(q) - 1) = 2\lg(\frac{q}{2})$ terms. □

4.2 Bloom filter construction

If the underlying system is based on Bloom filters, like SADS, we can describe the tradeoffs listed in the general construction in terms of increased Bloom filter size. The standard Bloom filter equation demands that in order to store n items with a false positive rate of p, our filter needs to have size m, show in formula (1). There are two factors that increase the necessary size of Bloom filters we must choose in order to maintain the same false positive rate. First, for every value inserted, there are now $d \lg r$ terms added, giving an increase in "effective" number of values for purposes of calculating proper sizes. Second, for every query, there are $2d \lg(\frac{q}{2})$ queries in CNF, any of which could be a false positive. If we assume in the worst case that a single false positive from a sub-query will cause a complete false positive, then we can give an upper bound on the multiplicative increase of false positive rate as $2d \lg(\frac{q}{2})$. Thus, the total size of the Bloom filter to ensure that the false positive rate does not exceed p is given by formula (2).

$$m = \frac{-n \ln p}{(\ln 2)^2} \text{ (1)} \qquad m = d \lg r \frac{-n \ln \frac{p}{2d \lg(\frac{q}{2})}}{(\ln 2)^2} \text{ (2)}$$

For practical purposes, this is a very reasonable increase in size, considering that most range query applications deal with orders of magnitude fewer values per record than exact string match queries, which may be used to index every single word in a tens-of-thousands-of-words long document.

An issue of greater concern is the magnification of existing privacy concerns, especially if we are using a system like SADS, which does not guarantee full protection of the result patterns. Because our construction will query the same sub-regions across multiple records in a query, and across multiple queries, if the result privacy is not protected against the server, he may be able to learn about the values stored within over time. For example, if a server sees that during a range query, two records had the same positive result for the same sub-region, it knows that they at the very least share a value in the same half-region (the largest possible sub-region). If over the course of multiple queries it sees those two documents match for a second sub-region, it then knows that they at the very least share a value in the same quarter-region. Over time, and seeing a sufficiently varied number of queries, it may learn exactly which documents share specific values. To prevent this, the ranged values could be re-indexed regularly based on the frequency of range queries being issued.

This is partially mitigated in the multi-dimensional case, since sub-regions of different dimensions cannot be differentiated, lending some additional obscurity. These are further obscured in systems like SADS where the ranged queries are interspersed with other types of queries including straight string matches and boolean queries. There is nothing to indicate to the holder of these identifiers what ranges they correspond to, or even if they are ranges at all. The quantitative evaluation of this reduction of information would depend on the nature of the records and their searchable attributes as well as the distribution of the queries that will be submitted. Therefore, the assessment of the significance of this leakage has to be done with respect to the specific data that will be used in the search scheme as well as the expected types of queries.

5. SADS SPECIFIC

5.1 Multiple Hash Functions

The BFs of different documents in the SADS scheme [30] share the same hash functions, and thus, the same BF indices for identical keywords. This is exploited by using a bit-slicing storage structure to improve query time. However, this has clear consequences for privacy:

• Due to commonality of indices for shared keywords, the search structures leak information to the IS about the similarity of the corresponding documents.

• The false positive rate of a single Bloom filter — the probability that a search query is matched incorrectly by it — with n bits, k hash functions, and m entries is $FP_{single} = (1 - (1 - \frac{1}{n})^{mk})^k$. If the false positive probabilities across different Bloom filters are independent, then the expected number of false positive results in a database with N documents is $FP_{single} \cdot N$. However, in the given situation, the false positive rates are not independent if documents share keywords. Let D_1 and D_2 be two documents where p fraction of the words in D_2 are also in D_1 and the query w is a false positive for the Bloom filter for D_1. The probability of a bit in BF_{D_2} to be set to 1 is $p + (1-p)(1-(1-\frac{1}{n})^{mk}))$ and therefore the probability D_2 has a false positive (all k search bits of w are set to 1) is $(p + (1-p)(1-(1-\frac{1}{n})^{mk})))^k$, which tends to 1 as p tends to 1.

We can avoid these issues by using different hash functions for the Bloom filters of each document. The BF indices for an entry would not be derived from its PH-DSAEP+ encryption but instead from keyed hashes of said encryption.

We implemented the *multiple hash functions* feature by generating a group of hash functions using a family of 2-universal hash functions [26]. In our implementation, we used HMAC over MD5 and SHA1 (using the document's ID as key) to generate BF hash functions, where the i-th hash function was $H_i(w) = H_1(w) + (i-1)H_2(w) \bmod P$, where P is a prime, $H_1(w)$ is HMAC(SHA1, ID, w), $H_2(w)$ is HMAC(MD5, ID, w) and w is the encrypted keyword.

5.2 Database Updates

So far we have assumed that the server's database does not change. It is preprocessed once in the beginning and from that point on the same data is used to answer all queries from the clients. However, in many practical situations the data of the server changes dynamically, which should be reflected correspondingly in the query results returned. The naive solution to run the preprocessing preparation of the database each time it changes, brings prohibitive efficiency cost. We would like to avoid processing each record in the database for updates that affect a small fraction of it. From a different point of view, though, the updates of the database can be considered private information of the server and thus the information about what records have been changed is a privacy leakage to any other party (in our case to the IS who holds the Bloom filter search structures). This type of leakage comes inherent with the efficiency requirement we posed above — if the update processing does not touch a record, clearly it has not been modified. Therefore, we accept the update pattern leakage as a necessary privacy trade-off for usable cost of the updates.

Now we look at the specific information that changes at the IS in the SADS scheme, and consider whether it has leakage beyond the update pattern of the documents:

• **Bloom filters**: As we discussed before, if we use the same hash function for the Bloom filters of all documents, then the search structures reveal the similarities between documents. In the case of an update this would be indicative to what fraction of the content of the document has been changed. If, however, each BF has a different set if hash functions, the update of a document would also include a selection of a new set of hash functions for its BF as well. The only information that the IS could derive from the update will be the change of the length of the document based on the number of 1's in the BF. However, this information can be obtained also from the length of the encrypted document that the IS is storing. In both cases, we can eliminate this leakage by padding the documents.

• **Encrypted documents** — Each encrypted document stored at the IS index is re-encrypted with a key that the IS obtains in an oblivious transfer execution with the data owner. If an existing document is modified, and the server encrypts it with the same key, then the IS can also use the same re-encryption key. If the server is adding a new document to his database, though, he should generate a new key of type k" (i.e., the encryption keys that the documents are encrypted with after the re-encryption of the IS). The guarantee that we want to provide is that the server does not know the permutation image of the key k'' that is used in the re-encryption of the document. We also want to avoid executing oblivious transfer for each document, which results in a complexity greater than the database size. Each permutation over $n + 1$ elements can be presented as the product of a permutation over the first n of the elements and a transposition of one of the n elements and the last unused element. Thus, it is sufficient to execute a protocol where the IS obtains re-encryption keys for the new document and for a random document from the rest. Intuitively this guarantees that the new re-encryption key could be assigned to any of the old documents or the new one, and if it was used for a previously existing document, then the new one receives the re-encryption key that was released from that document.

5.3 Optimizations

During the preprocessing stage, for each database document a Bloom filter containing its keywords is generated. In the SADS scheme, adding a keyword to the BF of a document involves encrypting the keyword under the server's key. Thus, preprocessing documents containing the same keyword incurs repeated effort. In order to avoid this unnecessary preprocessing cost, we can cache the BF indices for keywords. This avoids some recomputation, but requires additional storage space. Whether to do this, and how much to cache, depends on the nature of the documents and repeat frequency. This is also applicable in the case when multiple hash functions are used where the preprocessing of a keyword is not identical but shares a common and expensive intermediary result that can be reused. The caching capability we implement uses LRU removal policy.

In addition, SADS preprocesses each item of the dataset independently (i.e., computes the BF search structure for it), and furthermore, it handles the elements of each item separately (each word/value is inserted into the Bloom filer after a cryptographic transformation). This computational independence makes for simple and robust parallelization. The search phase, especially when using multiple hash functions, also permits parallelization of the computation of the search indices for a query. We used the open source Threading Building Blocks library [21] to implement the parallelization optimization. It is easy to use and well-integrated with C++. After analyzing the source code we found out that there is just one integer counter that we need to synchronize among the different threads: the Bloom filters counter. It took roughly 10 lines of code to parallelize the entire preprocessing phase – similar for the search phase too.

6. EVALUATION

To evaluate the practicality of our proposed extensions we implemented them in SADS (roughly 4 Klocs of C++ code in total) and we performed a number of measurements using realistic datasets: (i) the email dataset that was made public after the Enron scandal [31] and (ii) a synthetic dataset with personal information for 100K persons. The Enron dataset consists of about half a million emails with an average size of 900 bytes after stemming . During the preprocessing phase of SADS, a distinct Bloom filter for each email was created. Then, each of the email files was tokenized and the tokens where stored in the corresponding Bloom filter, after they were properly encrypted. The format of the second dataset is more close to a database than a collection of documents. Its schema consists of a single table with 51 attributes of three types: strings (first name, last name, etc.), numbers (height, SSN, etc.) and file links (fingerprint, private key, security image, etc.) and it is stored in a flat CSV (Comma Separated Value) file. The total size of that dataset, along with the files pointed in the records, is 51GB and the average size for a record is 512KB. During the preprocessing phase we created a distinct Bloom filter for each record and each of the attribute values where inserted after it was prefixed with the attribute name ("name_value") and properly encrypted. In both cases, we configured the BF parameters so as the false positive rate would be less than 10^{-6}.

The experimental evaluation setup was comprised by two servers and a client laptop. The servers had two four-quad Intel Xeon 2.5GHz CPUs, 16 GB of RAM, two 500 GB hard disk drives, and a 1 Gbit ethernet interface. The laptop was equipped with an Intel Core2 Duo 2.20GHz CPU, 4 GB of RAM, a 220 GB hard disk drive, and a 100 Mbit ethernet interface. All of them were connected through a Gigabit switch; they all ran a 64-bit flavor of the Ubuntu operating system. QR and IS were running on each of the servers, the queries were performed from the laptop. When Document Retrieval was enabled, the File Server was running on the same host with the IS.

6.1 Memory Consumption

Along with the timing measurements, we also monitored the memory consumption of the extended SADS system to determine scaling limits. We found out that the only significant factor was the type of Bloom filter storage. Bloom filters are stored either sequentially in a flat file or transposed using the slicing optimization. In the sequential storage case memory usage was constant; it grew consistently with the dataset size in the slicing case, because the structures are kept in memory and written to files at the end. During the search phase, both the client and the QR used a small, constant amount of memory ($\sim$2MB). On the other hand, the IS's memory usage grew with the dataset size. In the sequential storage case, the file was `mmap`'ed; the amount of memory used was the Bloom filter size in bytes times the number of BFs (e.g. 1KB * 50K = 50MB). When the slicing optimization was enabled, we saw higher memory usage, $\sim$109MB for the same dataset. That was most likely due to the extensive use of C++ vectors, which we can further optimize in the case of much larger databases where the available RAM may become an issue.

6.2 Implementation Optimizations

We performed experiments using variable-sized subsets of both datasets while changing the size of the cache. As for the Enron dataset, we show that a good cache size is 5K keywords. This gives us a $\sim$90% hit ratio, while reducing the preprocessing time for 50K emails from 2h to 10m. Performing the same experiments for the synthetic dataset yielded slightly worse results, as some attribute values are unique. However, using a 10K keywords cache the hit ratio was 50% on the full dataset, which still is a significant gain.

We measured the speedup of the preprocessing phase on the full datasets, while increasing the number of threads. As we expected, the speedup grew linearly until the number of threads reached the number of cores in our servers – that is eight. When the number of threads was more than the CPU cores, the speedup slightly de-

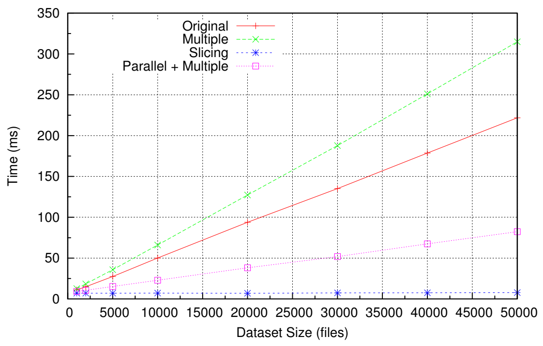


Figure 5: Average query time under different SADS configurations using the Enron dataset.

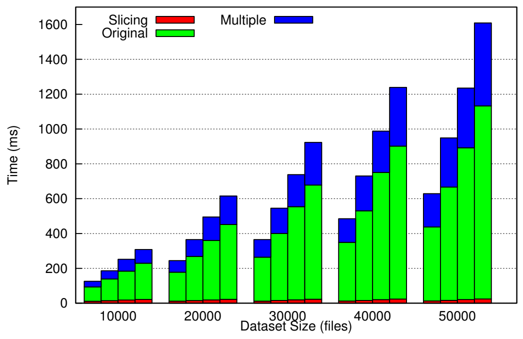


Figure 6: Average OR-query time under different SADS configurations using the Enron dataset. Each cluster is for a different dataset size and each bar is for a different term count (from 2 to 5).

clined, most probably due to thread scheduling overhead. Performance results for the parallelized search phase are presented in the next section.

6.3 Search Performance

The introduction of the multiple hash functions feature in SADS poses a trade-off between efficiency and privacy. Not only because of the higher computation overhead it adds but also because it is incompatible with the slicing optimization. In this section we explore in detail the effects of the multiple hash function scheme and also how parallel search could help amortize some of the performance penalty.

Figure 5 shows the comparison for four different configurations of SADS: (i) original, (ii) original with the slicing optimization enabled, (iii) using multiple hash functions and (iv) using multiple hash functions and parallel searching together. The search time reported in this figure is the total time elapsed from the point when the client issues the query to the QR until it receives the set of matching document IDs if any — no document retrieval. As expected, the average query time grows linearly using the original SADS configuration, as the actual search is done linearly over all the Bloom filters. Next, we can see that the slicing optimization greatly reduces search time to a point that it seems almost constant across different dataset sizes. Using the multiple hash functions feature we do get better privacy guarantees, but at the cost of increased search time by another factor that is proportional to the

Column	Statistics min/ avg/ max	Query Time msec (stdev)	# of Ranges # (stdev)
Age	2/ 38/ 95	1,529 (244)	4.3 (0.6)
Height	58/ 67/ 78	959 (123)	2.8 (0.4)
Weight	90/ 175/ 280	1,979 (345)	5.8 (0.9)
SSN	1M/3.9G/6.5G	12,783 (805)	38.0 (2.4)

* the time for a single keyword query is ~400 msec

Table 1: Range queries timing on integer attributes of the synthetic dataset (100K records – 51GB).

dataset size. That is because for each document we have to recalculate the hash functions and recompute the Bloom filter indices. Finally, we see that taking advantage of the commonly used multicore architectures does increase the performance of the search in the multiple hashing scheme. More precisely, the speedup when we used 8 threads on our 8-core servers was from 1.3 to almost 4 for the dataset sizes shown in the Figure 5. Thus, although the multiple hash functions feature increases the computation factor, we can amortize a great part of it by executing it in parallel. It is also worth noting that the multiple hash functions plus parallel searching configuration provides better performance that the original configuration, while on the same time it improves the privacy guarantees.

Next, we evaluate the performance overhead of the multiple hash functions in boolean queries, and more precisely OR queries. To optimize the normal case – i.e., when the slicing optimization is not enabled – we skip BFs that already contain one of the search terms. That way we avoid searching over and over on Bloom Filters that already match the OR query thus reducing the overall searching time, especially when the search terms are frequent. Figure 6 shows the search time for OR queries under different SADS configurations. Each cluster of bars is for a different dataset size; each bar is for a different term count in the boolean OR query. The first bar is for two terms, the second for three, and the last two for four and five, respectively. The fact that the search time in each cluster grows sub-linearly to the number of terms clearly shows the performance gain.

6.4 Range Queries

The implementation of the range queries extension on top of SADS translates a range to a variable-sized OR query. In the average case, the number of the terms in the OR query depends on the size of the numbers in the range and the size of the range itself. To evaluate the practicality of that approach, we measured the time for performing range queries over the numeric attributes of the synthetic dataset. These are *age*, *height*, *weight* and *SSN*. The range of the values of these attributes relative small, except for the SSN which spans from one million to a few billions (first column of Table 1). For each of the attributes, we calculated ten ranges that each match about 1/10 of the dataset. SADS was configured to use multiple hash functions and the parallel search optimization was enabled. Table 1 shows the average query time over the ten queries for each attribute, along with the average number of binary ranges that each query was translated to. In most of the cases, where the ranges are translated to a few binary ranges, the average range query time is low enough to be considered practical. On the other hand, the SSN attribute demonstrates the disadvantage of our range queries extension when dealing with big values. Still, the performance is not prohibitive, but, clearly, our range query extension yields better results for values that range from a few tens to a few thousands.

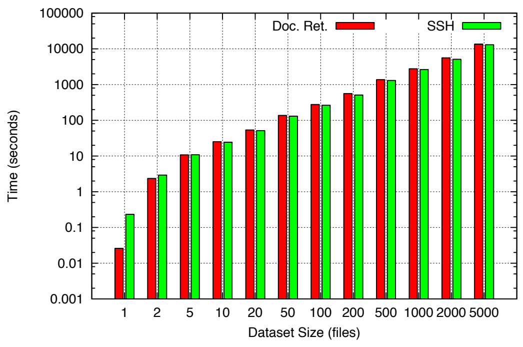


Figure 7: Average time for retrieving documents anonymously, compared to retrieving them non-anonymously using ssh file transfer. Average size of files being transferred was 27.8 Mb

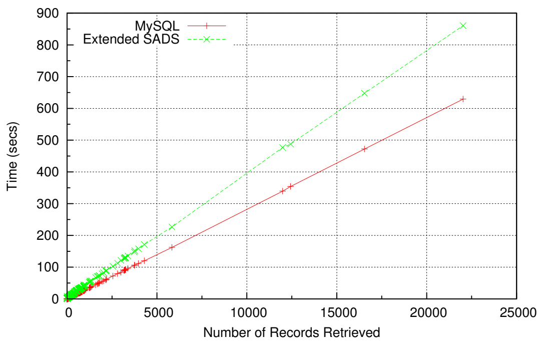


Figure 8: Comparison between the extended SADS and MySQL.

6.5 Document Retrieval

We implemented document retrieval using PH-SAEP and standard RSA signatures to sign query results. Using PH-SAEP puts a (likely over-restrictive) limit on the length of plaintext values. To handle this, we encrypt larger files using AES private key encryption, and store the key encrypted with PH-SAEP as a header in the encrypted file. The files can thus be read by decrypting the header with the appropriate PH-SAEP key and using the result to decrypt the content of the file. We preprocess the files in a way that provides an intermediate party with AES encrypted files under different AES keys and encryptions of these AES keys under some permutation of the keys k_1", $\ldots k_n$". The client will receive as results from the intermediary party the encrypted files, the encrypted AES keys, and the indices of the keys k" used for their encryptions. When he receives the decryption keys k" from the server, the client will first decrypt the AES keys and then use them to decrypt the remainder of the files.

Figure 7 shows the average time to retrieve documents using our scheme versus the number of documents being retrieved. This is shown in comparison to a non-privacy-preserving SSH-based file transfer. As we can see, our scheme adds very little overhead compared to the minimum baseline for encrypted transfer. The time also shows linear growth, suggesting that it is dominated by file encryption and transfer, rather than for the encryption and verification of the results vector itself.

As a point of comparison, Olumofin and Goldberg [29] present some of the best implementation performance results currently published for multi-selection single-server PIR. In their performance results, we see response times per retrieval ranging from 100 to 1000 seconds for retrievals of 5-10 documents on database sizes ranging from 1 to 28 GB. Our scheme scales strictly with number and size of documents retrieved, and not with the total database size. They do not state the sizes of the blocks retrieved in their scenario, but if we were to give a very high estimate of 1 MB per block, and assume they fetched 10 blocks every time, one could expect in our system that each query would take .7 seconds, still orders of magnitude short of the 100s fastest time they report for a 1GB database, and it would not scale up with increasing database size as theirs does, thus significantly beating the 1000s time they report for a 28GB database. Note that their system is not designed to protect privacy of the database, only of the request. The work of [14] presents a protocol for privacy-preserving policy-based information transfer, which achieves privacy guarantees weaker than SPIR and similar to ours. Direct comparison between our and their performance results is hard — they present timings only for the computation time without communication, which grows linearly with the size of their database. The maximum size of their database is 900 records with 2000 ms computation per record retrieval, while for our scheme the entire record retrieval time (computation plus communication) for a database with 25000 records is about 40ms (Figure 8, described in the next section).

6.6 Overall Performance

Finally, we compare the performance of the extended SADS system with a real world DBMS, MySQL. In order to do that, we implemented a SQL front-end for SADS that could parse simple conjunctive and disjunctive queries. Then, we loaded the synthetic dataset to both systems and we executed a number of queries of variable result set size. SADS was configured to use multiple hash functions and document retrieval was enabled. Parallel searching was also disabled, which means that we compared using the less efficient version of the extended SADS. Figure 8 shows the total duration of both the query and the retrieval of the data for our system and MySQL. Our scheme performs just 30% slower on average than MySQL, which is the price for the privacy guarantees it provides.

6.7 Case Study: Sharing of Health Records

We next examine, from a very high level, the suitability of our scheme for a hospital's admissions records database. (A database for full medical record storage is vastly more complex and is not addressed here.) A patient's health record is very sensitive information, usually kept by the patient's medical institution. There are cases, though, where such information needs to be shared among different institutions. For example, a patient may need to visit a different institution due to an emergency, or to get a second opinion. This sharing may need to be done privately for several reasons. In an emergency, a doctor may need to query all the institutions that share records with his without revealing the patient's identity, especially to the institutes that do not have information about him. If the querying is not private in that case, some institutions would learn information about a patient that has never visited them. Or, a patient may not want his institution to know which institution he visits for specialized needs, such as drug rehabilitation, so again the query for his record has to be performed privately.

A database of health records is similar to the synthetic dataset we used in our evaluation. It contains some searchable fields like name, date of birth, height, etc.; each record may be linked with several medical exam results like x-rays, electrocardiographs, magnetic tomographies, etc. In 1988, there were about ten routine tests during the hospital's admission process alone [20]; today, about thirty individual tests are done.[1] Taking into account that some of the results can be a few tens of Mbs — for example, a head CAT scan is about 32 MB — each health record could be a couple of hundred megabytes. One major hospital complex admits about 117K inpatients per year[2]; to a first approximation, their database would thus have several hundred thousands rows and 30–40 columns.

We have already seen, though, that the extended SADS scheme we propose can successfully handle a database of this size. Our evaluation demonstrated that document retrieval adds only a small overhead compared to simple transfer, thus easily scaling with the size of the document retrieved. Also, searching over 100K records with 51 searchable attributes each takes less than half a second, thus meeting real-world requirements. Finally, the support for updates in health records is a requirement covered by our extended SADS scheme. We conclude that our scheme is able to handle the requirements of this hospital, while preserving patient privacy.

7. RELATED WORK

Most of the existing constructions providing encrypted search capabilities aim to solve the case of database outsourcing [4–6, 8, 13, 33]. In this setting a party outsources the storage of his database to an untrusted server and wants to enable the server to execute searches on his behalf without learning information about either the data or the query. Unlike the data sharing scenario that we consider, this setting does not impose privacy requirements for the data with respect to the querier. A common technique in encrypted search schemes [4, 33] is to use trapdoors derived from query terms that enable the server to determine if a ciphertext matches the specific term. This implies the search complexity will be at best linear in the number of searchable tokens. A different approach in the setting of database outsourcing is to use inverted indices, where the search structures directly map all possible search terms to matches [8, 13]. Search then consists of finding the appropriate entry in the search structure for a given query's trapdoor. Such solutions leak the search pattern across a sequence of queries and are not easily extendable to allow more complicated queries beyond exact match when we need to preserve the privacy of the database from the querier.

Protecting the search pattern imposes efficiency costs. Bellare et al. [25] showed that in order to achieve sublinearity of the search complexity over encrypted ciphertexts, deterministic encryption is required, which leaks the search pattern. The works of [30] and [17] combine the idea of using deterministic encryption with Bloom filters [3] as search structures. However, the Bloom filter search structures constructed in these works leak the similarity of the underlying documents to the party who uses them for search. The work of [12] offers a scheme that exchanges search pattern leakage for efficiency improvement. While the suggested approach achieves sublinearity of the search complexity in terms of the number of searchable records, using preprocessing that transforms searchable tokens occurring in multiple records with unique tokens per record, it still requires time linear in the number of all searchable tokens contained in the matching records. Thus this solution is appropriate for scenarios with small numbers of searchable tokens per record, its efficiency improvements do not suffice in the case of long documents that contain many searchable keywords.

Search capability beyond simple exact matches has been achieved through constructions for attribute-based encryption [2, 19] and predicate encryption [22]. These approaches have a similar flavor to some of the searchable encryption schemes in the sense that they allow decryption only if the encrypted message satisfied a certain condition, which can be expressed, for example, as a dot product, Boolean formula or polynomial evaluation. But this also brings the related efficiency overhead that requires linearity in the size of all searchable tokens. Range queries are another type of queries with important practical applications. The work of [6] presents a protocol for range queries that comes with overhead $O(\sqrt{n})$ where n is the size of the domain of the searchable values. Shi et al. [32] incur $O((\log n)^D)$ computation overhead for D dimensional queries. Both of these schemes require that the token for the searchable interval is issued by the owner of the secret encryption key, which suffices for data outsourcing solutions but does not address the case of the data sharing.

If we consider the document retrieval functionality when the querier is the data owner, the search party can return the encrypted matching documents for which the querier has decryption keys. However, this approach is not applicable when the data owner and the querier are different parties and we want to hide from the data owner which documents were returned as results to the query. If the querier already knows the IDs of the documents of interest for his query the functionality of a symmetric private information retrieval (SPIR) scheme [16] when instantiated with the whole content of the database would theoretically allow the querier to retrieve the desired documents without the owner finding out what documents were retrieved. However, the efficiency cost is quite high. The PIR schemes that guarantee privacy for the query but do not provide privacy for the database already incur substantial efficiency overhead. Implementations of PIR were presented in [18, 29], and the work of [28] uses PIR techniques to provide partial privacy for SQL queries.

8. CONCLUSIONS

When we consider the question of secure search in practical settings, the privacy guarantees of a scheme are no longer the only relevant issue: a perfectly secure scheme that no one can use provides no actual privacy. The efficiency of an approach becomes a major factor in determining its usability given the available resources.

We adopted the relaxed security model of the SADS scheme; we extended its functionality by constructing a document retrieval protocol that runs in time proportional to the size of the returned set of documents and by providing range queries over integer data at a cost comparable to simple keyword queries in the average case. Both extensions take no advantage of any specific feature of SADS, making them applicable to any keyword-based private search system. Additionally, we improved SADS by: (i) providing a protocol that facilitates database updates without requiring processing of the whole database, (ii) using different hash functions for different BFs which provides better privacy guarantees and (iii) developing two implementation level optimizations, parallelization and caching.

The experimental results for the extended SADS system demonstrate its practicality: we achieve search and document retrieval time which is on the order of the time of ssh transfer and much better than the results from the most recent PIR implementation presented in [29] (note that the PIR protocol actually has weaker privacy guarantees than what we need since it does not provide database privacy), while we provide better privacy guarantees than the original SADS. In other words, we have provided strong-enough security and privacy, and at an acceptable cost.

[1]Private communication with a physician.
[2]`http://nyp.org/about/facts-statistics.html`

9. REFERENCES

[1] William Aiello, Yuval Ishai, and Omer Reingold. Priced oblivious transfer: How to sell digital goods. In *Proceedings of EUROCRYPT'01*, London, UK, 2001.

[2] John Bethencourt, Amit Sahai, and Brent Waters. Ciphertext-policy attribute-based encryption. In *Proceedings of S&P'07*, Washington, DC, USA, 2007.

[3] Burton H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 13(7):422–426, 1970.

[4] Dan Boneh, Giovanni Di Crescenzo, Rafail Ostrovsky, and Giuseppe Persiano. Public key encryption with keyword search. In *Proceedings of EUROCRYPT'04*, 2004.

[5] Dan Boneh, Eyal Kushilevitz, Rafail Ostrovsky, and William E. Skeith III. Public key encryption that allows PIR queries. In *Proceedings of CRYPTO'07*, 2007.

[6] Dan Boneh and Brent Waters. Conjunctive, subset, and range queries on encrypted data. In *Proceedings of TCC*. Springer, 2006.

[7] Ran Canetti. Security and composition of multiparty cryptographic protocols. *Journal of Cryptology*, 13, 2000.

[8] Yan cheng Chang and Michael Mitzenmacher. Privacy preserving keyword searches on remote encrypted data. In *Proceedings of ACNS*, volume 3531, 2005.

[9] Benny Chor, Niv Gilboa, and Moni Naor. Private information retrieval by keywords. Technical Report TR-CS0917, Dept. of Computer Science, 1997.

[10] Benny Chor, Oded Goldreich, Eyal Kushilevitz, and Madhu Sudan. Private information retrieval. *J. ACM*, 45(6):965–981, 1998.

[11] Giovanni Di Crescenzo, Tal Malkin, and Rafail Ostrovsky. Single database private information retrieval implies oblivious transfer. In *EUROCRYPT*, pages 122–138, 2000.

[12] Emiliano De Cristofaro, Yanbin Lu, and Gene Tsudik. Efficient techniques for privacy-preserving sharing of sensitive information. In *TRUST*, 2011.

[13] Reza Curtmola, Juan Garay, Seny Kamara, and Rafail Ostrovsky. Searchable symmetric encryption: improved definitions and efficient constructions. In *Proceedings of CCS'06*. ACM, 2006.

[14] Emiliano De Cristofaro, Stanislaw Jarecki, Jihye Kim, and Gene Tsudik. Privacy-preserving policy-based information transfer. In *Proceedings of PETS*, 2009.

[15] Craig Gentry and Zulfikar Ramzan. Single-database private information retrieval with constant communication rate. In *Proceedings of the 32nd International Colloquium on Automata, Languages and Programming*, 2005.

[16] Y. Gertner, Y. Ishai, E. Kushilevitz, and T. Malkin. Protecting data privacy in private information retrieval schemes. *Journal of Computer and System Sciences*, 60(3):592–629, 2000.

[17] Eu-Jin Goh. Secure indexes. Cryptology ePrint Archive, Report 2003/216, 2004. `http://eprint.iacr.org/2003/216/`.

[18] Ian Goldberg. Improving the robustness of private information retrieval. In *Proceedings of the IEEE Symposium on Security and Privacy*, 2007.

[19] Vipul Goyal, Abhishek Jain, Omkant Pandey, and Amit Sahai. Bounded ciphertext policy attribute based encryption. In *Proceedings of ICALP '08*, Berlin, Heidelberg, 2008.

[20] F. Allan Hubbell, Elizabeth B. Frye, Barbara V. Akin, and Lloyd Rucker. Routine admission laboratory testing for general medical patients. *Medical Care*, 26(6), 1988.

[21] Intel. Threading building blocks 2.2. http://www.threadingbuildingblocks.org/, 2009.

[22] J. Katz, A. Sahai, and B. Waters. Predicate encryption supporting disjunctions, polynomial equations, and inner products. In *Proceedings of EUROCRYPT*. Springer, 2008.

[23] Brian W. Kernighan and Rob Pike. *The Practice of Programming*. Addison-Wesley, 1999.

[24] Brian W. Kernighan and P.J. Plauger. *The Elements of Programming Style*. McGraw-Hill, 1974.

[25] A. Boldyareva M. Bellare and A. O'Neill. Deterministic and efficiently searchable encryption. In *Proceedings of CRYPTO'07*, 2007.

[26] Michael Mitzenmacher and Salil Vadhan. Why simple hash functions work: Exploiting the entropy in a data stream. In *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms*, pages 746–755, 2008.

[27] Moni Naor and Benny Pinkas. Efficient oblivious transfer protocols. In *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms*, pages 448–457, Philadelphia, PA, USA, 2001.

[28] Femi Olumofin and Ian Goldberg. Privacy-preserving queries over relational databases. In *Proceedings of PETS*, pages 75–92, 2010.

[29] Femi Olumofin and Ian Goldberg. Revisiting the computational practicality of private information retrieval. In *Financial Cryptography*, 2011.

[30] Mariana Raykova, Binh Vo, Steven Bellovin, and Tal Malkin. Secure anonymous database search. In *CCSW 2009.*, 2009.

[31] Jitesh Shetty and Jafar Adibi. The enron email dataset database schema and brief statistical report. Technical report, USC, 2004.

[32] Elaine Shi, John Bethencourt, T-H. Hubert Chan, Dawn Song, and Adrian Perrig. Multi-dimensional range query over encrypted data. In *Proceedings of S&P'07*, pages 350–364, Washington, DC, USA, 2007.

[33] Dawn Xiaodong Song, David Wagner, and Adrian Perrig. Practical techniques for searches on encrypted data. In *Proceedings of S&P'00*, Washington, DC, USA, 2000.

[34] Peter Williams and Radu Sion. Usable PIR. In *NDSS*, 2008.

The Socialbot Network: When Bots Socialize for Fame and Money

Yazan Boshmaf, Ildar Muslukhov, Konstantin Beznosov, Matei Ripeanu
University of British Columbia
Vancouver, Canada
{boshmaf,ildarm,beznosov,matei}@ece.ubc.ca

ABSTRACT

Online Social Networks (OSNs) have become an integral part of today's Web. Politicians, celebrities, revolutionists, and others use OSNs as a podium to deliver their message to millions of active web users. Unfortunately, in the wrong hands, OSNs can be used to run astroturf campaigns to spread misinformation and propaganda. Such campaigns usually start off by infiltrating a targeted OSN on a large scale. In this paper, we evaluate how vulnerable OSNs are to a large-scale infiltration by *socialbots*: computer programs that control OSN accounts and mimic real users. We adopt a traditional web-based botnet design and built a *Socialbot Network* (SbN): a group of adaptive socialbots that are orchestrated in a command-and-control fashion. We operated such an SbN on Facebook—a 750 million user OSN—for about 8 weeks. We collected data related to users' behavior in response to a large-scale infiltration where socialbots were used to connect to a large number of Facebook users. Our results show that (1) OSNs, such as Facebook, can be infiltrated with a success rate of up to 80%, (2) depending on users' privacy settings, a successful infiltration can result in privacy breaches where even more users' data are exposed when compared to a purely public access, and (3) in practice, OSN security defenses, such as the Facebook Immune System, are not effective enough in detecting or stopping a large-scale infiltration as it occurs.

1. INTRODUCTION

Online Social Networks (OSNs) such as Facebook[1] and Twitter[2] have far exceeded the traditional networking service of connecting people together. With millions of users actively using their platforms, OSNs have attracted third parties who exploit them as an effective media to reach and potentially influence a large and diverse population of web users [21, 23]. For example, during the 2008 U.S. presidential election, social media was heavily employed by Obama's campaign team who raised about half a billion dollars online, introducing a new digital era in presidential fundraising [40]. Moreover, it has been argued that OSNs, as democracy-enforcing communication platforms, were one of the key enablers of the recent Arab Spring in the Middle East [35, 38]. Such a global integration of social media into everyday life is rapidly becoming the norm, and arguably is here to stay [8]. But what if some of the content in social media—OSNs in particular—is not written by human beings?

A new breed of computer programs called *socialbots* are now online, and they can be used to influence OSN users [24]. A socialbot is an automation software that controls an account on a particular OSN, and has the ability to perform basic activities such as posting a message and sending a connection request. What makes a socialbot different from self-declared bots (e.g., Twitter bots that post up-to-date weather forecasts) and spambots is that it is designed to be stealthy, that is, it is able to pass itself off as a human being. This allows the socialbot to compromise the social graph of a targeted OSN by infiltrating (i.e., connecting to) its users so as to reach an influential position. This position can be then exploited to spread misinformation and propaganda in order to bias the public opinion [26]. For example, Ratkiewicz et al. [33] describe the use of Twitter bots to run astroturf and smear campaigns during the 2010 U.S. midterm elections.

As socialbots infiltrate a targeted OSN, they can further harvest private users' data such as email addresses, phone numbers, and other personal data that have monetary value. To an adversary, such data are valuable and can be used for online profiling and large-scale email spam and phishing campaigns [30]. It is thus not surprising that different kinds of socialbots are being offered for sale in the Internet black-market for as much as $29 per bot [4].

Recently, many techniques have been proposed that aim to automatically identify spambots in OSNs based on their abnormal behavior [31, 16, 37]. For example, Stein et al. [36] present the Facebook Immune System (FIS): an adversarial learning system that performs real-time checks and classification on every read and write action on Facebook's database, all for the purpose of protecting its users and the social graph from malicious activities. It is, however, not well-understood how such defenses stand against socialbots that mimic real users, and what the expected users' behavior might be in response to a large-scale infiltration by such bots.

In this paper, we aim to fill this knowledge gap. We treat large-scale infiltration in OSNs as an organized campaign that is run by an army of socialbots to connect to either random or targeted OSN users on a large scale. Therefore,

[1]`http://www.facebook.com`
[2]`http://www.twitter.com`

we decided to adopt a traditional web-based botnet design and define what we call a *Socialbot Network* (SbN): a group of re-programmable socialbots that are orchestrated by an adversary (called a *botherder*) using a software controller (called a *botmaster*). The botmaster is designed to exploit the known properties of social networks, such as the *triadic closure principle* [32], and use them as heuristics to define commands, which increase the magnitude of the potential infiltration in the targeted OSN.

We built a simple, yet effective, SbN consisting of 102 socialbots and a single botmaster. We then operated this SbN on Facebook for 8 weeks. During that time, the socialbots were able to send a total of 8,570 connection requests. We recorded all data related to the anticipated infiltration and the corresponding users' behavior, along with all accessible users' profile information. Overall, we summarize our main findings as follows:

(1) OSNs, such as Facebook, are highly vulnerable to a large-scale infiltration. From the OSN side, we show that it is not difficult to fully automate the overall operation of an SbN, including accounts creation. From the users' side, we show that most OSN users are not careful enough when accepting connection requests sent by strangers, especially when they have mutual connections. This behavior can be exploited to achieve a large-scale infiltration with a success rate of up to 80%.

(2) Depending on users' privacy settings, operating an SbN can result in many privacy breaches. We show that greater number of users' data can be harvested after a large-scale infiltration. This data include email addresses, phone numbers, and other profile information, all of which have monetary value. Unfortunately, this also includes the private data of users who have not been infiltrated, but are connected to infiltrated users. Moreover, we show that a botherder can operate an SbN conservatively, at a slow pace, and still collect an average of 175 new chunks of publicly-unaccessible users' data per socialbot per day.

(3) In practice, OSN security defenses such as the FIS are not effective enough in detecting a large-scale infiltration. Our results show that a successful infiltration of an OSN user is expected to be observed within the first 3 days after the request has been sent by a socialbot. This means that the social graph will rapidly change in a relatively short time, and the socialbots will get gradually integrated into the targeted online community. We found that the FIS was able to block only 20% of the accounts used by the socialbots. This, however, was a result of feedback from users who flagged these accounts as spam. In fact, we did not observe any evidence that the FIS detected what was really going on: an organized large-scale infiltration campaign.

The rest of the paper is organized as follows: We first provide background information and define our notations in Section 2. After that, we present the concept of a Socialbot Network, along with its design goals and construction details, in Section 3. Next, we demonstrate our experiments with an SbN on Facebook in Section 4, and then we discuss our results in Section 5. This is followed by an outline of related works in Section 6. Finally, we conclude the paper in Section 7.

2. PRELIMINARIES

In what follows, we present background information and define the notations we use in the upcoming discussion.

2.1 Online Social Networks

Online Social Networks (OSNs) provide centralized web platforms that facilitate users' social activities. A user in such a platform owns an account and is represented by a profile that describes her social attributes such as name, gender, interests and contact information. We use the terms "account", "profile", and "user" interchangeably. A social connection between two users can be either undirected like friendships in Facebook, or directed like follower-followee relationships in Twitter.

An OSN can be modeled as a graph $G = (V, E)$, where V represents a set of users and E represents a set of social connections among these users. For every user $u \in V$, the set $\Gamma(u)$ is called the *neighborhood* of u, and it contains all users in V with whom u has social connections. We denote the average neighborhood size in G by $N_{\texttt{avg}} = |V|^{-1} \sum_{u \in V} |\Gamma(u)|$. Finally, we call the set $\Delta(u) = \bigcup_{v \in \Gamma(u)} \Gamma(v)$ the *extended neighborhood* of u.

2.2 Social Engineering and Socialbots

Traditionally, *social engineering* is defined as the art of gaining access to secure objects by exploiting human psychology, rather than using hacking techniques. Social engineering, however, has become more technical and complex; social engineering attacks are being computerized and fully automated, and are becoming adaptive and context-aware [9, 5]. In fact, some of these attacks are sophisticated and use learned or hard-coded heuristics and observations about users' behaviour in the targeted system so as to increase the magnitude of their potential damage [5, 6, 20].

The next generation of social engineering attacks is even more deceptive; they employ an automation software called a *socialbot* that controls a profile in an OSN, and has the ability to execute basic online social activities. For example, Realboy [10] is an experimental project that aims to design believable Twitter bots that imitate real Twitter users.

2.3 OSN Vulnerabilities

We discuss four vulnerabilities found in today's OSN which allow an adversary to carry out a large-scale infiltration campaign. We treat each vulnerability separately and provide evidence to support it.

2.3.1 Ineffective CAPTCHAs

OSNs employ CAPTCHAs [42] to prevent automated bots from abusing their platforms. An adversary, however, can often circumvent this countermeasure by using different techniques such as automated analysis using optical character recognition [6], exploiting botnets to trick infected victims into manually solving CAPTCHAs [5, 12], reusing session IDs of known CAPTCHAs [18], cracking MD5 hashes of CAPTCHAs that are validated at the client side [44], and hiring cheap human labor [27].

Let us consider the use of cheap human labor to break CAPTCHAs; a phenomenon that is known as CAPTCHA-breaking business. Motoyama et al. [27] show that companies involved in such a business are surprisingly efficient; they have high service quality with a success rate of up to 98%, charge $1 per 1,000 successfully-broken CAPTCHAs, and provide software APIs to automate the whole process. Thus, even the most sophisticated CAPTCHA technology that only humans could solve can be effectively circumvented with a small investment from an adversary. In such a situation, the adversary acts as an economist; he would invest in such businesses if the return on investment is considerably high. This allows researchers to look at online attacks from an economic context, and define cost metrics that measure when it is economically feasible for an adversary to mount a large-scale attack that involves, for instance, breaking CAPTCHAs through employing cheap human labor [17].

2.3.2 Fake User Accounts and Profiles

Creating a user account on an OSN involves three tasks: providing an active email address, creating a user profile, and sometimes solving a CAPTCHA. Each user account maps to one profile, but many user accounts can be owned by the same person or organization using different email addresses. In what follows, we argue that an adversary can fully automate the account creation process. This, however, is not new, as similar tools are used for online marketing [2].

Fake user accounts: When creating a new user account in an OSN, an email address is required to first validate and then activate the account. The OSN validates the account by associating it to the owner of the email address. After account validation, its owner activates the account by following an activation link that is emailed by the OSN. Accordingly, an adversary has to overcome two hurdles when creating a new account: providing an active email address that he owns, and account activation. To tackle the first hurdle, the adversary can maintain many emails by either using "temp" email addresses that are obtained from providers that do not require registration such as `10minutemail.com`, or by creating email addresses using email providers that do not limit the number of created emails per browsing session or IP address such as `mail.ru`. As for the second hurdle, an adversary can write a simple script that downloads the activation email, and then sends an HTTP request to the activation URL that is included in the downloaded email.

Fake user profiles: Creating a user profile is a straightforward task for real users; they just have to provide the information that represents their social attributes. For an adversary, however, the situation is a bit different. The objective of the adversary is to create profiles that are "socially attractive". We consider a purely adversarial standpoint concerning social attractiveness; the adversary aims to exploit certain social attributes that have shown to be effective in getting users' attention. Such attributes can be inferred from recent social engineering attacks. Specifically, using a profile picture of a good looking woman or man has had the greatest impact [6, 14]. Thus, an adversary can use publicly available personal pictures for the newly created profiles, with the corresponding gender and age-range. In fact, the adversary can use already-rated personal pictures from websites like `hotornot.com`, where users publicly post their personal pictures for others to rate their "hotness".[3] It is thus possible for an adversary to automate the collection of the required profile information through crawling (or scavenging in this case) the Web.

[3]Such sites also provide categorization of the rated personal pictures based on gender and age-range.

2.3.3 Crawlable Social Graphs

The social graph of an OSN is usually hidden from public access in order to protect its users' privacy. An adversary, however, can reconstruct parts of the social graph by first logging in to the OSN platform using an account, and then traversing through linked user profiles starting from a "seed" profile. In the second task, web crawling techniques can be used to download profile pages and then scrape their content. This allows the adversary to parse the connections lists of user profiles, such as the "friends list" in Facebook, along with their profile information. After that, the adversary can gradually construct the corresponding social graph with all accessible social attributes using a breadth-first search [25]. The adversary can build either a customized web crawler for this task or resort to cheap commercial crawling services that support social-content crawling such as `80legs.com`.

2.3.4 Exploitable Platforms and APIs

Most OSNs provide software APIs that enable the integration of their platforms into third-party software systems. For example, Facebook Graph API [1] enables third parties to read from and write data into Facebook, and provides a simple and consistent view of Facebook's social graph by uniformly representing objects (e.g., profiles, photos) and the connections between them (e.g., friendships, likes, tags). An adversary, however, can use such APIs to automate the execution of online social activities. If an activity is not supported by the API, then the adversary can scrape the content of the platform's web pages, and record the exact HTTP requests which are used to carry out such an activity (i.e., HTTP-request templates). In particular, sending connection requests is often not supported, and is protected against automated usage by CAPTCHAs. This is also the case if a user sends too many requests in a short time period. An adversary, however, can always choose to reduce the frequency at which he sends the requests to avoid CAPTCHAs. Another technique is to inject artificial connection requests into normal OSN traffic at the HTTP level, so that it would appear as if the users added the adversary as a friend [19].

3. THE SOCIALBOT NETWORK

We first start with a conceptual overview of a Socialbot Network (SbN), and briefly outline the adversarial objectives behind maintaining such a network. This is followed by a discussion on the SbN design goals, after which we outline its construction details.

3.1 Overview

We define a *Socialbot Network* (SbN) as a set of socialbots that are owned and maintained by a human controller called the *botherder* (i.e., the adversary). An SbN consists of three components: socialbots, a botmaster, and a Command & Control (C&C) channel. Each socialbot controls a profile in a targeted OSN, and is capable of executing commands that result in operations related to social interactions (e.g., posting a message) or the social structure (e.g., sending a connection request). These commands are either sent by the botmaster or predefined locally on each socialbot. All data collected by the socialbots are called the *botcargo* and are always sent back to the botmaster. A *botmaster* is an OSN-independent software controller that the botherder interacts with in order to define and then send commands through the C&C channel. The C&C *channel* is a communication

Table 1: The generic operations supported by a socialbot in any given OSN.

Operation	Type	Description
`read`(o, p)	Social-interaction	Reads an object o from profile p and returns its value v as botcargo
`write`(v, o, p)	Social-interaction	Writes value v to object o on profile p
`connect`(b, p)	Social-structure	Sends or accepts a connection request sent from profile b to profile p
`disconnect`(b, p)	Social-structure	Breaks the social connection between profiles b and p

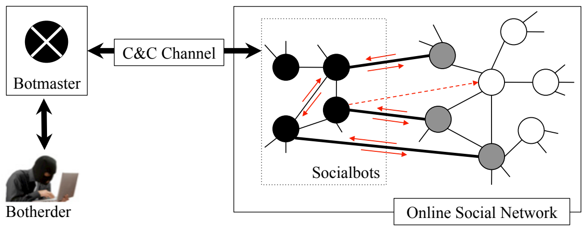


Figure 1: A Socialbot Network. Each node in the OSN represents a profile. The socialbots are marked in black. Infiltrated profiles are marked in gray. Edges between nodes represent social connections. The dashed arrow represent a connection request. The small arrows represent social interactions.

channel that facilitates the transfer of both the botcargo and the commands between the socialbots and the botmaster, including any heartbeat signals. Figure 1 shows a conceptual model of an SbN.

3.2 Objectives

The botherder is a person or an organization that builds and operates an SbN for two main objectives: (1) to carry out a large-scale infiltration campaign in the targeted OSN, and (2) to harvest private users' data. The first objective involves connecting to a large number of either random or targeted OSN users for the purpose of establishing an influential position or fame. The second objective, on the other hand, aims to generate profit by collecting personal users' data that have monetary value. Notice that this data can be then used to craft personalized messages for subsequent spam, phishing, or astroturf campaigns.

3.3 Design Goals

Ideally, an SbN has to be fully automated and scalable enough to control hundreds of socialbots. This is achieved by adopting a traditional web-based botnet design. In order to be effective, however, an SbN has to meet three challenging goals: (1) each socialbot has to be designed in such a way that hides its true face; a robot, (2) the botmaster has to implement heuristics that enable large-scale infiltration in the targeted OSN, and (3) the traffic in the C&C channel has to look benign in order to avoid detecting the botmaster.

In this paper, we decided to use a simplistic design in order to meet each one of these goals. We used techniques that have shown to be both feasible and effective. We discuss the details of these techniques in the following section.

3.4 Construction

We now discuss how a botherder can construct an SbN that performs well in practice while meeting the design goals outlined in the previous section.

3.4.1 The Socialbots

A socialbot consists of two main components: a profile on a targeted OSN (the face), and the socialbot software (the brain). We enumerate the socialbots with the profiles they control, that is, for a set $\mathcal{B} = \{b_1, \ldots, b_n\}$ of n socialbots, we use $b_i \in \mathcal{B}$ to refer to both the i-th socialbot and the profile it controls. But how should the socialbot software be programmed in order to mimic real users?

First, we require the socialbot to support two types of generic operations in any given OSN: social-interaction operations that are used to read and write social content, and social-structure operations that are used to alter the social graph. A description of these operations is shown in Table 1.

Second, we define a set of commands that each includes a sequence of generic operations. Each command is used to mimic a real user action that relates to social content generation (e.g., a status update) or social networking (e.g., joining a community of users). Commands can be either defined locally on each socialbots (called *native commands*), or sent by the botmaster through the C&C channel (called *master commands*). For example, we define a native command called `status_update` as follows: at arbitrary times, a socialbot $b_i \in \mathcal{B}$ generates a message m (e.g., a random blurb crawled from the Web), and executes the operation `write`(m, o, b_i) where o is the object that maintains messages on profile b_i (e.g., the profile's "wall" in Facebook).

Finally, each socialbot employs a *native controller*: a simple two-state Finite-State Machine (FSM) that enables the socialbot to either socialize by executing commands, or stay dormant.

3.4.2 The Botmaster

A botmaster is a botherder-controlled automation software that orchestrates the overall operation of an SbN. The botmaster consists of three main components: a botworker, a botupdater, and a C&C engine. The *botworker* builds and maintains socialbots. Building a new socialbot involves first creating a new socially attractive profile in the targeted OSN as discussed in Section 2.3.2. After that, the profile's credentials (i.e., the user name and password) are delegated to the socialbot software so as to get a full control over this profile. The *botupdater* pushes new software updates, such as a new list of native commands, to the socialbots through the C&C channel. Finally, the C&C *engine* maintains a repository of master commands and runs a *master controller*: a many-state FSM that is the core control component of the SbN. The botherder interacts with the C&C engine to define a set of master commands, which are dispatched when needed by the master controller and then sent to the socialbots. An interesting question now follows: what kinds of master commands are required to achieve a large-scale infiltration in the targeted OSN?

First, notice that at the beginning each socialbot is isolated from the rest of the OSN, that is, $|\Gamma(b_i)| = 0$ for each

Table 2: Master commands. The socialbot $b_i \in \mathcal{B}$ is the socialbot executing the command, $|\mathcal{B}| = n$.

Command	Description
`cluster`	Connects b_i to at most $N_{\texttt{avg}}$ other socialbots in $\mathcal{B}$
`rand_connect(`k`)`	Connects b_i to k non-boherder-owned profiles that are picked at random from the OSN
`decluster`	Disconnects b_i from every socialbot $b_j \in \mathcal{S}$ where $\mathcal{S} = \{b_j \mid b_j \in \Gamma(b_i) \cap \mathcal{B}$ and $|\Gamma(b_j)| > n\}$
`crawl_extneighborhood`	Returns $\Delta(b_i)$, the extended neighborhood of b_i, as botcargo
`mutual_connect`	Connects b_i to every profile $p_j \in \Delta(b_i) - \mathcal{B}$.
`harvest_data`	Reads all accessible information of every profile $p_j \in \Gamma(b_i)$, and returns it as botcargo

$b_i \in \mathcal{B}$, which is not a favorable structure to start a large-scale infiltration. Tong et al. [39] show that the social attractiveness of a profile in an OSN is highly correlated to its neighborhood size, where the highest attractiveness is observed when the neighborhood size is close to the network's average. Usually, $N_{\texttt{avg}}$ is known or can be estimated (e.g., $N_{\texttt{avg}} = 130$ on Facebook [3]). Thus, in order to increase the social attractiveness of a socialbot, the adversary defines a master command `cluster`, which orders each socialbot to connect to at most $N_{\texttt{avg}}$ other socialbots.

Second, it has been widely observed that if two users have a mutual connection in common, then there is an increased likelihood that they become connected themselves in the future [22]. This property is known as the *triadic closure principle*, which originates from real-life social networks [32]. Nagle et al. [29] show that the likelihood of accepting a connection request in an OSN is about three times higher given the existence of some number of mutual connections. Therefore, in order to improve the potential infiltration in the targeted OSN, the adversary defines a master command `mutual_connect`, which orders each socialbot to connect to user profiles with whom it has mutual connections.

Finally, we design the master controller to switch between three master states or phases: setup, bootstrapping, and propagation. In the *setup* phase, the botmaster builds n socialbots, updates their software, and then issues the `cluster` command. After that, in the *bootstrapping* phase, the botmaster issues the command `rand_connect(`k`)`, which orders each socialbot to connect to k profiles that are picked at random from the targeted OSN. When every socialbot is connected to k non-botherder-owned profiles, the botmaster issues the command `decluster`, which orders the socialbots to break the social connections between them, and hence, destroying any n-clique structure that could have been created in the earlier step. In the *propagation* phase, the botmaster issues the command `crawl_extneighborhood`, which orders the socialbots to crawl their extended neighborhoods, after which the botmaster uses this information and issues the command `mutual_connect`. Whenever a socialbot infiltrates a user profile, the botmaster issues the command `harvest_data`, which orders the socialbot to collect all accessible users' profile information in its neighborhood. A description of all master commands is shown in Table 2.

3.4.3 *The C&C Channel*

The communication model of an SbN consists of two channels: the C&C channel and the socialbot-OSN channel. The socialbot-OSN channel carries only OSN-specific API calls and normal HTTP traffic, which are the end product of executing a command by a socialbot. From the OSN side, this traffic originates from either an HTTP proxy in case of high activity, or from a normal user. It is therefore quite difficult to identify a socialbot solely based on the traffic in the socialbot-OSN channel.

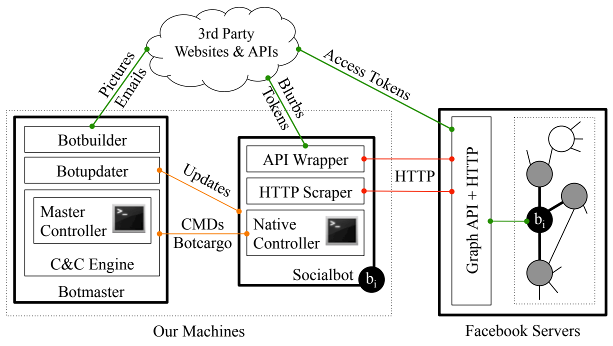


Figure 2: The Facebook Socialbot Network.

As for the C&C channel, how should it be built so that it is particularly hard to identify the botmaster? To start with, we argue that detecting the botmaster from the C&C traffic is as hard as it is in a traditional botnet; the botherder can rely on the existing botnet infrastructure and deploy the SbN as part of the botnet. Alternatively, the botherder can employ advanced techniques that, for example, establish a probabilistically unobservable communication channel by building a covert OSN botnet [28].

4. EVALUATION

In order to evaluate how vulnerable OSNs are to a large-scale infiltration by an SbN, we decided to build one according to the discussion in Section 3.4. We chose Facebook as a targeted OSN because we believe it is particularly difficult to operate an SbN in Facebook for the following reasons: (1) unlike other OSNs, Facebook is mostly used to connect to offline friends and family but not to strangers [13], and (2) Facebook employs the Facebook Immune System (FIS): an adversarial learning system which represents a potential nemesis of any SbN [36].

4.1 Ethics Consideration

Given the nature of an SbN, a legitimate question follows: is it ethically acceptable and justifiable to conduct such a research experiment? We believe that minimal-risk realistic experiments are the only way to reliably estimate the feasibility of an attack in real-world. These experiments allow us, and the wider research community, to get a genuine insight into the ecosystem of online attacks, which are useful in understanding how similar attacks may behave and how to defend against them. This seems to be the opinion of other researchers who share our belief [6, 20].

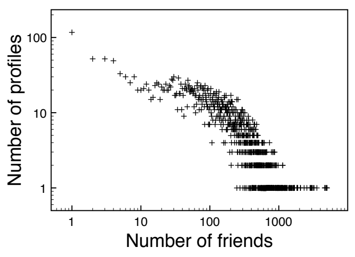


Figure 3: Degree distribution of the generated random sample of Facebook user profiles.

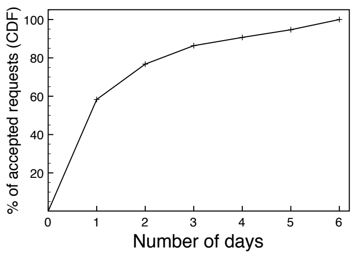


Figure 4: Cumulative distribution of number of days and accepted friendship requests.

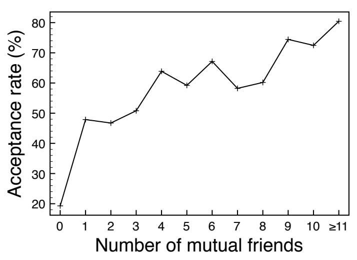


Figure 5: Overall infiltration as a function of number of mutual friends.

We carefully designed our experiment in order to reduce any potential risk at the user side by following known practices [7], and got the approval of our university's behavioral research ethics board. We strongly encrypted and properly anonymized all collected data, which we have completely deleted after we finished our planned data analysis.

4.2 The Facebook SbN

Figure 2 shows the architecture of the SbN we developed. Each socialbot ran the same software and was equipped with only one native command; `status_update`. We implemented the generic operations described in Table 1 using two techniques: API calls and HTTP-request templates, which we now briefly describe. First, we exploited Facebook's Graph API [1] to carry out the social-interaction operations. The API, however, requires the user (i.e., the socialbot in this case) to be logged in to Facebook at the time of any API call. To avoid this, we developed a Facebook application that fetches permanent OAuth 2.0 access tokens that allow each socialbot to send API calls without the need to login. Second, for the social-structure operations, we used pre-recorded HTTP-request templates that allow each socialbot to send friendship requests as if they were sent from a browser. We used an API provided by `iheartquotes.com` to pull random quotes and blurbs which we used as messages for the status updates. As for the botmaster software, we implemented the botworker to interface with three useful websites: `decaptcher.com`; a CAPTCHA-breaking business, `hotornot.com`; a photo-sharing website, and `mail.ru`; an email provider. We also implemented the botupdater with an enhanced functionality to update the HTTP-request templates, along with any new native commands. Finally, we implemented all master commands described in Table 2.

The master command `rand_connect` requires some extra attention. On Facebook, each profile has a unique ID that is represented by a 64-bit integer and is assigned at the time the profile is created. In order to get a uniform sample of Facebook profiles, we decided to use a simple random sampling technique called *rejection sampling* [34], which we now descirbe. First, we generated 64-bit integers at random, but with a range that is reduced to the known ID ranges used by Facebook [15]. Next, we tested whether each generated ID mapped to a real profile by probing the profile page using this ID. Finally, if the profile existed, we included the profile ID in the random sample only if this profile was not isolated. We define an *isolated* user profile as a profile that does not display its "friends list" or has no friends of Facebook.

We deployed the simple two-state native controller and the three-phase, many-state master controller. We acknowledge, however, that more sophisticated controllers could be used that, for instance, employ some machine learning algorithms in order to improve the potential infiltration.

4.3 Operating the Facebook SbN

We operated the Facebook SbN for 8 weeks. The socialbots were able to send a total of 8,570 friendship requests, out of which 3,055 requests were accepted by the infiltrated users. We divide the following discussion according to the three phases of the master controller.

4.3.1 Setup

We created 102 socialbots and a single botmaster, all of which are physically hosted on one machine for simplicity. A botherder, however, could resort to a more sophisticated deployment such as a P2P overlay network. Even though we could have built the socialbots automatically using the botworker, we decided to create them manually as we had no intention to support any CAPTCHA-breaking business. In total, we created 49 socialbots that had male user profiles (referred to as m-socialbots), and 53 socialbots that had female user profiles (referred to as f-socialbots).

4.3.2 Bootstrapping

The socialbots generated a random sample of $5,053$ valid profile IDs. These IDs passed the inclusion criteria we discussed in Section 4.2. Figure 3 shows the degree distribution of this sample.[4]

Based on a pilot study, we decided to send 25 friendship requests per socialbot per day in order to avoid CAPTCHAs. The socialbots took 2 days to send friendship requests to all of the $5,053$ profiles. In total, exactly $2,391$ requests were sent from m-socialbots and $2,662$ from f-socialbots. We kept monitoring the status of the requests for 6 days. Overall, 976 requests were accepted with an average acceptance rate of 19.3%. In particular, 381 of the accepted requests were sent from m-socialbots (15.9% acceptance rate), and 595 were sent from f-socialbots (22.3% acceptance rate). About 86% of the infiltrated profiles accepted the requests within the first three days of the requests being sent, as shown in Figure 4. Overall, the SbN spent two weeks in the bootstrapping phase. For most of that time, however, the SbN was setting idle.

[4]The *degree* of a node is the size of its neighborhood, and the *degree distribution* is the probability distribution of these degrees over the whole network (or a sample of it).

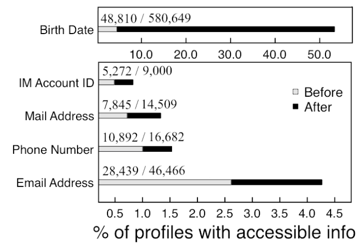


Figure 6: Data revelation of selected profile info before and after a large-scale infiltration.

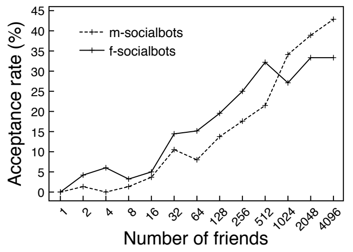


Figure 7: Infiltration as a function of number of friends a user profile has.

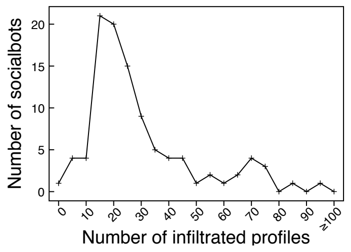


Figure 8: The distribution of number of infiltrated profiles among the socialbots.

4.3.3 Propagation

We kept the SbN running for another 6 weeks. During this time, the socialbots added 3,517 more user profiles from their extended neighborhoods, out of which 2,079 profiles were successfully infiltrated. This resulted in an average acceptance rate of 59.1%, which, interestingly, depends on how many mutual friends the socialbots had with the infiltrated users, and can increase up to 80% as shown in Figure 5.

By the end of the eighth week, we decided to take the SbN down as it resulted in a heavy traffic with Facebook. In total, the SbN generated approximately 250GB inbound and 3GB outbound traffic. We consider the operation time a conservative estimate of the real performance of the SbN as we paused it several times for debugging and data analysis, especially during the bootstrapping phase. We believe that operating the SbN for a longer time is expected to increase the average acceptance rate as the propagation phase will have a higher contribution.

4.4 Harvested Data

As the socialbots infiltrated Facebook, they harvested a large set of users' data. We were able to collect news feeds, users' profile information, and "wall" messages. We decided, however, to only focus on users' data that have monetary value such as Personally Identifiable Information (PII).

After excluding all remaining friendships between the socialbot, the total size of all direct neighborhoods of the socialbots was 3,055 profiles. The total size of all extended neighborhoods, on the other hand, was as large as 1,085,785 profiles. In Table 3, we compare users' data revelation of some PII before and after operating the SbN, as a percentage of the neighborhoods size.

To emphasize its significance, we visualize the data revelation difference of selected profile information in Figure 6. We include all user profiles from *both* the direct and the extended neighborhoods of the socialbots, which added up to 1,088,840 profiles. Each bar in the figure is annotated with two numbers in x/y format, where x and y represent the number of profiles with accessible profile information before and after a large-scale infiltration, respectively.

5. DISCUSSION

In what follows, we discuss the results presented in the previous section and focus on four main points: the observed users' behavior, the effectiveness of the Facebook Immune System, the infiltration performance of the socialbots, and the expected implications on other software systems.

Table 3: Data revelation as % of neighborhoods size.

Neighborhoods	**Direct(%)**		**Extended(%)**	
Profile Info	**Before**	**After**	**Before**	**After**
Gender	69.1	69.2	84.2	84.2
Birth Date	03.5	72.4	04.5	53.8
Married To	02.9	06.4	03.9	04.9
Worked At	02.8	04.0	02.8	03.2
School Name	10.8	19.7	12.0	20.4
Current City	25.4	42.9	27.8	41.6
Home City	26.5	46.2	29.2	45.2
Mail Address	00.9	19.0	00.7	01.3
Email Address	02.4	71.8	02.6	04.1
Phone Number	00.9	21.1	01.0	01.5
IM Account ID	00.6	10.9	00.5	00.8
Average	13.3	34.9	15.4	23.7

5.1 Users' Behavior

Given the results presented in Section 4, someone might ask: are the infiltrated profiles real after all, or are they just other socailbots? To begin with, notice that during the bootstrapping phase, the socialbots targeted profiles that were picked at random out of millions of user profiles, and thus, it is highly unlikely to have picked mostly socialbots.

We also support this argument by the following analysis of the observed users' behavior. First of all, consider Figure 5. The big jump in the acceptance rate from users who were picked at random to those with whom the socialbots had some mutual friends is expected. It directly exhibits the effect of the triadic closure principle, which predicts that having mutual friends will improve the liklihood of accepting a friendship request as discussed in Section 3.4.2. The triadic closure, interestingly, also operated from the users side; the socialbots received a total of 331 friendship requests from their extended neighborhoods.

Second, the behavior depicted in Figure 4 matches the official statistics about real users on Facebook: 50% of the 750 million active Facebook users log on in any given day [3], and thus, it is expected that approximately half of the accepted friendship requests are observed within one day of the requests being sent.

Third and last, the users who were infiltrated during the bootstrapping phase, that is, those who were selected at random, showed another expected behavior [39]: the more friends they had, the higher the chance was that they accepted a friendship request from a socialbot (i.e., a stranger), as shown in Figure 7.

Table 4: SbN *new* data collection performance.

# Profiles Per	Request	Bot	Bot, Day
Gender	0.00	0.05	0.00
Birth Date	62.06	5,214.11	93.11
Married To	1.33	111.58	1.99
Worked At	0.42	35.54	0.63
School Name	10.66	896.05	16.00
Current City	17.61	1,479.87	26.43
Home City	20.29	1,704.99	30.45
Mail Address	0.78	65.33	1.17
Email Address	2.10	176.74	3.16
Phone Number	0.68	56.76	1.01
IM Account ID	0.44	36.55	0.65
Total	116.37	9,777.57	174.60

5.2 SbN vs. Facebook Immune System

The Facebook Immune System (FIS) is a coherent, scalable, and extensible real-time adversarial learning system that is deployed by Facebook to protect its users and the social graph from malicious activities [36]. It performs real-time checks and classification on every read and write action on Facebook database. An interesting question now follows: how did the FIS perform against the SbN we have operated?

After operating the SbN for 8 weeks, only 20 profiles used by the socialbots were blocked by the FIS, and interestingly, all of them were controlled by f-socialbots. After further investigation, we found that these profiles were blocked because some Facebook users flagged them as spam.[5] In fact, we did not observe any evidence that the FIS detected what was really going on other than relying on users' feedback, which seems to be an essential but potentially dangerous component of the FIS.

In reaction, we asked ourselves: what assumptions are made by the FIS that might be problematic? The answer came directly from the authors of the FIS: they state that "fake accounts have limited virality because they are not central nodes in the graph and lack trusted connections. They also have no unique data or history" [36]. Hence, we conjecture that the FIS does not consider fake accounts as a real threat. Fake accounts, however, are one of the main OSN vulnerabilities that allow a botherder to run a large-scale infiltration campaign. Detecting and blocking such accounts—as early as possible—is the main challenge that OSN security defenses like the FIS have to overcome in oder to win the battle against an SbN.

Finally, we noticed that employing the commands `cluster` and `status_update`, which we describe in Table 2, had a desirable effect. It appears that the socialbots seemed more human-like as only 20% of the 102 profiles they controlled got blocked, as apposed to 93% of the 15 profiles we used in our pilot study where we decided not to use these commands. This, in a sense, reflects one of the drawbacks of relying on users' feedback.

5.3 Infiltration Performance

One way to judge whether the resulting infiltration was the making of a small number of "outstanding" socialbots is to compare them in terms of the number of profiles they have infiltrated, as shown in Figure 8. Accordingly, we group the socialbots into two leagues representing the two humps in the figure. The first one constitutes 86% of the socialbots and 70% of the overall infiltration, where each socialbot has infiltrated 0–50 user profiles. The second league, on the other hand, constitutes 10% of the socialbots and 23% of the overall infiltration, where each socailbot has infiltrated 60–80 user profiles. The rest of the socialbots, which we consider as outliers, constitute only 4% of the socialbots with 7% of the overall infiltration.

As most of the resulted infiltration was the outcome of the socialbots' group work, we decided to calculate how much *new* data a botherder can collect per socialbot, as opposed to a completely public access. This is particularly useful when trying to estimate how many socialbots (or connection requests) are needed in order to collect a targeted amount of specific users' data, such as email addresses and birth dates. Using a conservative SbN operation of 25 friendship requests per socialbot per day, and a relaxed operation time of 8 weeks, we found that a botherder is expected to collect an average of 175 new chunks of users' data in Facebook per socialbot per day, as shown in Table 4.

5.4 Implications on other Systems

Operating the SbN for an extended period of time is expected to result in a significantly larger infiltration, as the socialbots will spend most of that time in the propagation phase. Accordingly, the neighborhood size of each socialbot is expected to keep increasing. This, however, has alarming implications on software systems that use the social graph of an OSN to provide services such as limiting Sybil nodes in distributed systems [45] and modeling trust in socially-informed recommender systems [43].

To explain why this is particularly important, let us consider OSN-based Sybil defenses used in P2P overlay networks.[6] In general, such a defense mechanism uses two types of networks that share the same nodes: the P2P network which we try to protect, and an external social network representing a trust network such as Facebook [45]. Now, in order to detect Sybil nodes in the P2P network, the following assumption is often made [41]: a Sybil node cannot have many connections with non-Sybil nodes in the social network. Thus, finding a well-connected node in the P2P network, which is loosely connected in the social network, is a good indication that this node is a Sybil. This assumption, however, is not safe. We showed that the socialbots can have arbitrarily many social connections with arbitrary OSN users. Therefore, we believe that using a social network such as Facebook to detect Sybil nodes in P2P networks is, in fact, ineffective. This conclusion extends to all systems that make this assumption about OSNs.

6. RELATED WORK

The closest threat to large-scale infiltration in OSNs is a botnet called Koobface [5]. At first, Koobface compromises user accounts on OSNs, and then uses these accounts to promote a provocative message with a hyperlink. The link points to a phishing website that asks the user to install

[5]Based on the content of a pop-up message that Facebook showed when we manually logged in.

[6]A Sybil node (or a Sybil for short) is an adversary-owned online identity, account, profile, or endpoint in a particular network. Sybil nodes represent a serious threat and can be used to subvert services such routing in P2P networks [11].

a Flash plugin which is, in fact, the Koobface executable. Unlike Koobface, an SbN does not rely on hijacked profiles as doing so requires infecting a large number of initial "zombie" machines through OSN-independent distribution channels.

Bilge et al. [6] show that most users in OSNs are not cautious when accepting connection requests that are sent to them. The authors did an experiment to test how willing users are to accept connection requests from forged user profiles of people who were already in their friendship list as confirmed contacts. They also compared that with users' response to connection requests from people that they do not know (i.e., fake profiles representing strangers). In their experiment, they show that the acceptance rate for forged profiles was always over 60%, and about 20% for the fake profiles. Unlike their targeted attack, we do not expect the adversary to forge profiles as this limits the scalability of the attack and makes it more susceptible to detection.

Other than the wide botnet literature, the majority of the work we found relates to Twitter bots and different techniques to detect them. Ratkiewicz et al. [33] describe the use of Twitter bots to spread misinformation in the run-up to the U.S. political elections. Stringhini et al. [37] analyze to what extent spam has entered OSNs, and how spammers who target these OSNs operate. The authors develop techniques to detect spammers, and show that it is possible to automatically identify the accounts they use. Grier et al. [16] show that 8% of the unique URLs that are sent in Twitter are later blacklisted. The authors also described a test on Tweets' timestamps to identify automated Twitter accounts, or spambots, by regularities in the times when they tweet, which they have found to have a high accuracy in identifying Twitter spammers.

7. CONCLUSION

We have evaluated how vulnerable OSNs are to a large-scale infiltration by a Socialbot Network (SbN). We used Facebook as a representative OSN, and found that using bots that mimic real OSN users is effective in infiltrating Facebook on a large scale, especially when the users and the bots share mutual connections. Moreover, such socialbots make it difficult for OSN security defenses, such as the Facebook Immune System, to detect or stop an SbN as it operates. Unfortunately, this has resulted in alarming privacy breaches and serious implications on other socially-informed software systems. We believe that large-scale infiltration in OSNs is only one of many future cyber threats, and defending against such threats is the first step towards maintaining a safer social Web for millions of active web users.

8. ACKNOWLEDGMENTS

We would like to thank San-Tsai Sun, Elizeu Santos-Neto, Albina Muslukhova, and Bader AlAhmad for their kind help and advice. We also would like to thank Cormac Herley, Miranda Mowbray, and Adriana Iamnitchi for their feedback on an early draft of this paper. This research is partially supported through funding from the NSERC Internetworked Systems Security Network (ISSNet).

9. REFERENCES

[1] Facebook Open Graph Protocol. `http://developers.facebook.com/docs/opengraph`.

[2] Sick profile maker. `http://sickmarketing.com/sick-profile-maker`.

[3] Facebook Statistics. `http://www.facebook.com/press`, March 2011.

[4] Jet bots. `http://allbots.info`, 2011.

[5] J. Baltazar, J. Costoya, and R. Flores. The real face of Koobface: The largest web 2.0 botnet explained. *Trend Micro Research*, July 2009.

[6] L. Bilge, T. Strufe, D. Balzarotti, and E. Kirda. All your contacts are belong to us: automated identity theft attacks on social networks. In *WWW '09: Proceedings of the 18th International Conference on World Wide Web*, pages 551–560, New York, NY, USA, 2009. ACM.

[7] N. Bos, K. Karahalios, M. Musgrove-Chávez, E. S. Poole, J. C. Thomas, and S. Yardi. Research ethics in the facebook era: privacy, anonymity, and oversight. In *CHI EA '09: Proceedings of the 27th international conference extended abstracts on Human factors in computing systems*, pages 2767–2770, New York, NY, USA, 2009. ACM.

[8] D. Boyd. Social media is here to stay... Now what? *Microsoft Research Tech Fest*, February 2009.

[9] G. Brown, T. Howe, M. Ihbe, A. Prakash, and K. Borders. Social networks and context-aware spam. In *CSCW '08: Proceedings of the ACM 2008 Conference on Computer Supported Cooperative Work*, pages 403–412, New York, NY, USA, 2008. ACM.

[10] Z. Coburn and G. Marra. Realboy: Believable twitter bots. `http://ca.olin.edu/2008/realboy`, April 2011.

[11] J. R. Douceur. The sybil attack. In *IPTPS '01: Revised Papers from the First International Workshop on Peer-to-Peer Systems*, pages 251–260, London, UK, 2002. Springer-Verlag.

[12] M. Egele, L. Bilge, E. Kirda, and C. Kruegel. Captcha smuggling: hijacking web browsing sessions to create captcha farms. In *SAC '10: Proceedings of the 2010 ACM Symposium on Applied Computing*, pages 1865–1870, New York, NY, USA, 2010. ACM.

[13] N. B. Ellison, C. Steinfield, and C. Lampe. The benefits of Facebook "friends:" social capital and college students' use of online social network sites. *Journal of Computer-Mediated Communication*, 12(4):1143–1168, July 2007.

[14] N. FitzGerald. New Facebook worm - don't click da' button baby! `http://fitzgerald.blog.avg.com/2009/11/`, November 2009.

[15] M. Gjoka, M. Kurant, C. T. Butts, and A. Markopoulou. Walking in Facebook: A case study of unbiased sampling of osns. In *Proceedings of IEEE INFOCOM '10*, San Diego, CA, March 2010.

[16] C. Grier, K. Thomas, V. Paxson, and M. Zhang. @spam: the underground on 140 characters or less. In *Proceedings of the 17th ACM Conference on Computer and Communications Security*, CCS '10, pages 27–37, New York, NY, USA, 2010. ACM.

[17] C. Herley. The plight of the targeted attacker in a world of scale. In *The 9th Workshop on the Economics of Information Security (WEIS 2010)*, 2010.

[18] C. Hernandez-Castro and A. Ribagorda. Remotely

telling humans and computers apart: An unsolved problem. In J. Camenisch and D. Kesdogan, editors, *iNetSec 2009 ? Open Research Problems in Network Security*, volume 309 of *IFIP Advances in Information and Communication Technology*, pages 9–26. Springer Boston, 2009.

[19] M. Huber, M. Mulazzani, and E. Weippl. Who on earth is Mr. Cypher? automated friend injection attacks on social networking sites. In *Proceedings of the IFIP International Information Security Conference 2010: Security & Privacy — Silver Linings in the Cloud*, 2010.

[20] T. N. Jagatic, N. A. Johnson, M. Jakobsson, and F. Menczer. Social phishing. *Commun. ACM*, 50(10):94–100, 2007.

[21] A. M. Kaplan and M. Haenlein. Users of the world, unite! the challenges and opportunities of social media. *Business Horizons*, 53(1):59 – 68, 2010.

[22] J. Leskovec and E. Horvitz. Planetary-scale views on a large instant-messaging network. In *Proceeding of the 17th International Conference on World Wide Web*, pages 915–924, New York, NY, USA, 2008. ACM.

[23] G. Livingston. Social media: The new battleground for politics. `http://mashable.com/2010/09/23/congress-battle-social-media/`, September 2010.

[24] D. Misener. Rise of the socialbots: They could be influencing you online. `http://www.cbc.ca/news/technology/story/2011/03/29/f-vp-misener-socialbot-armies-election.html`, March 2011.

[25] A. Mislove, M. Marcon, K. P. Gummadi, P. Druschel, and B. Bhattacharjee. Measurement and analysis of online social networks. In *IMC '07: Proceedings of the 7th ACM SIGCOMM Conference on Internet Measurement*, pages 29–42, New York, NY, USA, 2007. ACM.

[26] E. Morozov. Swine flu: Twitter's power to misinform. `http://neteffect.foreignpolicy.com/posts/2009/04/25/swine_flu_twitters_power_to_misinform`, April 2009.

[27] M. Motoyama, K. Levchenko, C. Kanich, D. McCoy, G. M. Voelker, and S. Savage. Re: Captchas: understanding captcha-solving services in an economic context. In *Proceedings of the 19th USENIX Conference on Security*, USENIX Security'10, pages 28–28, Berkeley, CA, USA, 2010. USENIX Association.

[28] S. Nagaraja, A. Houmansadr, P. Agarwal, V. Kumar, P. Piyawongwisal, and N. Borisov. Stegobot: A covert social network botnet. In *Proceedings of the Information Hiding Conference*, 2011.

[29] F. Nagle and L. Singh. Can friends be trusted? exploring privacy in online social networks. In *Proceedings of the 2009 International Conference on Advances in Social Network Analysis and Mining*, pages 312–315, Washington, DC, USA, 2009. IEEE Computer Society.

[30] S. Patil. Social network attacks surge. `http://www.symantec.com/connect/blogs/social-network-attacks-surge`, June 2011.

[31] A. Pitsillidis, K. Levchenko, C. Kreibich, C. Kanich, G. M. Voelker, V. Paxson, N. Weaver, and S. Savage. Botnet judo: Fighting spam with itself. In *NDSS*, 2010.

[32] A. Rapoport. Spread of information through a population with socio-structural bias: I. assumption of transitivity. *Bulletin of Mathematical Biology*, 15:523–533, 1953. 10.1007/BF02476440.

[33] J. Ratkiewicz, M. Conover, M. Meiss, B. Gonçalves, S. Patil, A. Flammini, and F. Menczer. Truthy: mapping the spread of astroturf in microblog streams. In *Proceedings of the 20th international conference companion on World wide web*, WWW '11, pages 249–252, New York, NY, USA, 2011. ACM.

[34] C. P. Robert and G. Casella. *Monte Carlo Statistical Methods (Springer Texts in Statistics)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2005.

[35] F. Salem and R. Mourtada. Civil movements: The impact of Facebook and Twitter. *The Arab Social Media Report*, 1(2), 2011.

[36] T. Stein, E. Chen, and K. Mangla. Facebook immune system. In *Proceedings of the 4th Workshop on Social Network Systems*, SNS '11, pages 8:1–8:8, New York, NY, USA, 2011. ACM.

[37] G. Stringhini, C. Kruegel, and G. Vigna. Detecting spammers on social networks. In *Proceedings of the 26th Annual Computer Security Applications Conference*, ACSAC '10, pages 1–9, New York, NY, USA, 2010. ACM.

[38] C. Taylor. Why not call it a Facebook revolution? `http://edition.cnn.com/2011/TECH/social.media/02/24/facebook.revolution/`, February 2011.

[39] S. T. Tong, B. Van Der Heide, L. Langwell, and J. B. Walther. Too much of a good thing? the relationship between number of friends and interpersonal impressions on Facebook. *Journal of Computer-Mediated Communication*, 13(3):531–549, 2008.

[40] J. A. Vargas. Obama raised half a billion online. `http://voices.washingtonpost.com/44/2008/11/obama-raised-half-a-billion-on.html`, November 2008.

[41] B. Viswanath, A. Post, K. P. Gummadi, and A. Mislove. An analysis of social network-based sybil defenses. In *Proceedings of the ACM SIGCOMM 2010 conference on SIGCOMM*, SIGCOMM '10, pages 363–374, New York, NY, USA, 2010. ACM.

[42] L. von Ahn, M. Blum, N. J. Hopper, and J. Langford. Captcha: Using hard AI problems for security. In E. Biham, editor, *EUROCRYPT*, volume 2656 of *Lecture Notes in Computer Science*, pages 294–311. Springer, 2003.

[43] F. Walter, S. Battiston, and F. Schweitzer. A model of a trust-based recommendation system on a social network. *Autonomous Agents and Multi-Agent Systems*, 16:57–74, 2008.

[44] H. Yeend. Breaking CAPTCHA without OCR. `http://www.puremango.co.uk/2005/11/breaking_captcha_115/`, November 2005.

[45] H. Yu, P. B. Gibbons, M. Kaminsky, and F. Xiao. Sybillimit: A near-optimal social network defense against sybil attacks. In *Proceedings of the 2008 IEEE Symposium on Security and Privacy*, pages 3–17, Washington, DC, USA, 2008. IEEE Computer Society.

Detecting and Resolving Privacy Conflicts for Collaborative Data Sharing in Online Social Networks

Hongxin Hu, Gail-Joon Ahn and Jan Jorgensen
Arizona State University
Tempe, AZ 85287, USA
{hxhu,gahn,jan.jorgensen}@asu.edu

ABSTRACT

We have seen tremendous growth in online social networks (OSNs) in recent years. These OSNs not only offer attractive means for virtual social interactions and information sharing, but also raise a number of security and privacy issues. Although OSNs allow a single user to govern access to her/his data, they currently do not provide any mechanism to enforce privacy concerns over data associated with multiple users, remaining privacy violations largely unresolved and leading to the potential disclosure of information that at least one user intended to keep private. In this paper, we propose an approach to enable collaborative privacy management of shared data in OSNs. In particular, we provide a systematic mechanism to identify and resolve privacy conflicts for collaborative data sharing. Our conflict resolution indicates a tradeoff between privacy protection and data sharing by quantifying privacy risk and sharing loss. We also discuss a proof-of-concept prototype implementation of our approach as part of an application in Facebook and provide system evaluation and usability study of our methodology.

Categories and Subject Descriptors

D.4.6 [**Security and Protection**]: Access controls; H.2.7 [**Information Systems**]: Security, integrity, and protection

General Terms

Security, Management

Keywords

Social Networks, Collaborative, Data Sharing, Privacy Conflict, Access Control

1. INTRODUCTION

Online social networks (OSNs), such as Facebook, Twitter, and Google+, have become a *de facto* portal for hundreds of millions of Internet users. For example, Facebook, one of representative social network provider, claims that it has more than 800 million active users [3]. With the help of these OSNs, people share personal and public information and make social connections with friends, coworkers, colleagues, family and even with strangers. As a result, OSNs store a huge amount of possibly sensitive and private information on users and their interactions. To protect that information, privacy control has been treated as a central feature of OSNs [2, 4].

OSNs provide built-in mechanisms enabling users to communicate and share information with other members. A typical OSN offers each user with a virtual space containing profile information, a list of the user's friends, and web pages, such as *wall* in Facebook, where the user and friends can post content and leave messages. A user profile usually includes information with respect to the user's birthday, gender, interests, education and work history, and contact information. In addition, users can not only upload a content into their own or others' spaces but also *tag* other users who appear in the content. Each tag is an explicit reference that links to a user's space. For the protection of user data, current OSNs indirectly require users to be system and policy administrators for regulating their data, where users can restrict data sharing to a specific set of trusted users. OSNs often use *user relationship* and *group membership* to distinguish between trusted and untrusted users. For example, in Facebook, users can allow *friends*, *friends of friends*, *specific groups* or *everyone* to access their data, relying on their personal privacy requirements.

Despite the fact that OSNs currently provide privacy control mechanisms allowing users to regulate access to information contained in their *own* spaces, users, unfortunately, have no control over data residing *outside* their spaces [7, 15, 21, 22, 24]. For instance, if a user posts a comment in a friend's space, s/he cannot specify which users can view the comment. In another case, when a user uploads a photo and tags friends who appear in the photo, the tagged friends cannot restrict who can see this photo. Since multiple associated users may have different privacy concerns over the shared data, *privacy conflicts* occur and the lack of collaborative privacy control increases the potential risk in leaking sensitive information by friends to the public.

In this paper, we seek an effective and flexible mechanism to support privacy control of shared data in OSNs. We begin by giving an analysis of data sharing associated with multiple users in OSNs, and articulate several typical scenarios of privacy conflicts for understanding the risks posed by those conflicts. To mitigate such risks caused by privacy conflicts, we develop a collaborative data sharing mechanism to support the specification and enforcement of multiparty privacy concerns, which have not been accommodated by existing access control approaches for OSNs (e.g., [10, 12, 13]). In the meanwhile, a systematic conflict detection and resolution mechanism is addressed to cope with privacy conflicts occurring in collaborative management of data sharing in OSNs. Our conflict resolution approach balances the need for privacy protection and the users' desire for information sharing by quantitative analysis of privacy risk and sharing loss. Besides, we implement a proof-of-concept prototype of our approach in the context of Facebook. Our experimental results based on comprehensive system evaluation and usability study demonstrate the feasibility and practicality of our solution.

The rest of the paper is organized as follows. In Section 2, we analyze several conflict scenarios for privacy control in OSNs. In Section 3, we address our proposed mechanism for detecting and resolving privacy conflicts in collaborative data sharing. The details on our prototype implementation and experimental results are described in Section 4. Section 5 gives a brief overview of related work. Section 6 concludes this paper and discusses our future directions.

2. PRIVACY CONFLICTS IN ONLINE SOCIAL NETWORKS

Users in OSNs can post statuses and notes, upload photos and videos in their own spaces, tag others to their content, and share the content with their friends. On the other hand, users can also post content in their friends' spaces. The shared content may be connected with multiple users. Consider an example where a photograph contains three users, Alice, Bob and Carol. If Alice uploads it to her own space and tags both Bob and Carol in the photo, we called Alice the *owner* of the photo, and Bob and Carol *stakeholders* of the photo. All of them may be desired to specify privacy policies to control over who can see this photo. In another case, when Alice posts a note stating *"I will attend a party on Friday night with @Carol"* to Bob's space, we call Alice the *contributor* of the note and she may want to make the control over her notes. In addition, since Carol is explicitly identified by *@-mention* (at-mention) in this note, she is considered as a *stakeholder* of the note and may also want to control the exposure of this note. Since each associated user may have different privacy concerns over the shared content, privacy conflicts can occur among the multiple users.

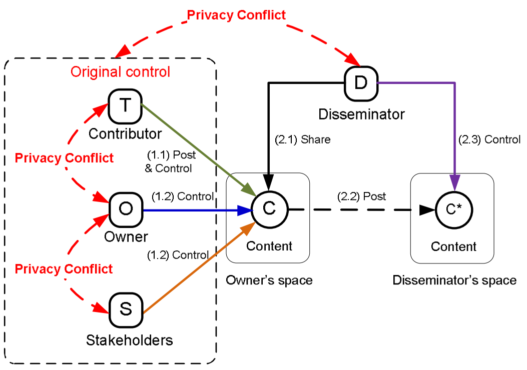


Figure 1: Privacy Conflicts in OSNs.

OSNs also enable users to share others' content. For example, when Alice views a photo in Bob's space and decides to share this photo with her friends, the photo will be in turn posted to her space and she can authorize her friends to see this photo. In this case, Alice is a *disseminator* of the photo. Since Alice may adopt a weaker control saying the photo is visible to everyone, the initial privacy concerns of this photo may be violated, resulting in the leakage of sensitive information during the procedure of data dissemination. Figure 1 shows a comprehensive conflict scenario in content sharing where the sharing starts with a *contributor* who uploads the content, and then a disseminator views and shares the content. All privacy conflicts among the *disseminator* and the original controllers (the *owner*, the *contributor* and the *stakeholders*) should be taken into account for regulating access to content in disseminator's space.

In addition to privacy conflicts in *content sharing*, conflicts may also occur in two other situations, *profile sharing* and *friendship sharing*, where multiple parties may have different privacy requirements in sharing their profiles and friendship lists with others or social applications in OSNs.

3. OUR APPROACH

Current online social networks, such as Facebook, only allow the data *owner* to fully control the shared data, but lack a mechanism to specify and enforce the privacy concerns from other associated users, leading to privacy conflicts being largely unresolved and sensitive information being potentially disclosed to the public. In this section, we address a collaborative privacy management mechanism for the protection of shared data with respect to multiple controllers in OSNs. A privacy policy scheme is first introduced for the specification and enforcement of multiparty privacy concerns. Then, we articulate our systematic method for identifying and resolving privacy conflicts derived from multiple privacy concerns for collaborative data sharing in OSNs.

3.1 Collaborative Control for Data Sharing in OSNs

3.1.1 *OSN Representation*

An OSN can be represented by a friendship network, a set of user groups and a collection of user data. The friendship network of an OSN is a graph, where each node denotes a user and each edge represents a friendship link between two users. Besides, OSNs include an important feature that allows users to be organized in groups [25, 26], where each group has a unique name. This feature enables users of an OSN to easily find other users with whom they might share specific interests (e.g., same hobbies), demographic groups (e.g., studying at the same schools), political orientation, and so on. Users can join in groups without any approval from other group members. Furthermore, OSNs provide each member a web space where users can store and manage their personal data including profile information, friend list and content. We now provide an abstract representation of an OSN with the core components upon which to build our solution:

- U is a set of users of the OSN, $\{u_1, \dots, u_n\}$. Each user has a unique identifier;
- G is a set of groups to which the users can belong, $\{g_1, \dots, g_m\}$. Each group also has a unique identifier;
- $UU \subseteq U \times U$ is a binary user-to-user friendship relation;
- $UG \subseteq U \times G$ is a binary user-to-group membership relation;
- P is a collection of user profile sets, $\{p_1, \dots, p_o\}$, where $p_i = \{p_{i1}, \dots, p_{ip}\}$ is the profile of a user $i \in U$. Each profile entry is a <*attribute: profile-value*> pair, $p_{ij} =< attr_j : pvalue_j >$, where $attr_j$ is an attribute identifier and $pvalue_j$ is the attribute value;
- F is a collection of user friend sets, $\{f_1, \dots, f_q\}$, where $f_i = \{u_1, \dots, u_r\}$ is the friend list of a user $i \in U$;
- C is a collection of user content sets, $\{c_1, \dots, c_s\}$, where $c_i = \{c_{i1}, \dots, c_{it}\}$ is a set of content of a user $i \in U$, where c_{ij} is a content identifier; and

- D is a collection of data sets, $\{d_1, \ldots, d_u\}$, where $d_i = p_i \cup f_i \cup c_i$ is a set of data of a user $i \in U$.

3.1.2 Privacy Policy Specification

To enable a collaborative management of data sharing in OSNs, it is essential for privacy policies to be in place to regulate access over shared data, representing privacy requirements from multiple associated users. Recently, several access control schemes (e.g., [9, 12]) have been proposed to support fine-grained privacy specifications for OSNs. Unfortunately, these schemes can only allow a single user to specify her/his privacy concern. Indeed, a flexible privacy control mechanism in a multi-user environment like OSNs should allow multiple controllers, who are associated with the shared data, to specify privacy policies.

Controller Specification: As we discussed previously in the privacy conflict scenarios (Section 2), in addition to the *owner* of data, other controllers, including the *contributor*, *stakeholder* and *disseminator* of data, also need to regulate the access of the shared data. We define these controllers as follows:

DEFINITION 1. *(Owner). Let $e \in d_u$ be a data item in the space of a user $u \in U$ in the social network. The user u is called the owner of e, denoted as OW_e^u.*

DEFINITION 2. *(Contributor). Let $e \in d_{u'}$ be a data item published by a user $u \in U$ in the space of another user $u' \in U$ in the social network. The user u is called the contributor of e, denoted as CB_e^u.*

DEFINITION 3. *(Stakeholder). Let $e \in d_{u'}$ be a data item in the space of a user $u' \in U$ in the social network. Let G be the set of tagged users associated with e. A user $u \in U$ is called a stakeholder of e, denoted as ST_e^u, if $u \in G$.*

DEFINITION 4. *(Disseminator). Let $e \in d_{u'}$ be a data item shared by a user $u \in U$ from the space of another user $u' \in U$ to her/his space in the social network. The user u is called a disseminator of e, denoted as DS_e^u.*

Then, we can formally define the controller specification as follows:

DEFINITION 5. *(Controller Specification). Let $cn \in U$ be a user who can regulate the access of data. And let $ct \in CT$ be the type of the cn, where $CT = \{OW, CB, ST, DS\}$ is a set of controller types, indicating Owner, Contributor, Stakeholder and Disseminator, respectively. The controller specification is defined as a tuple $< cn, ct >$.*

Accessor Specification: Accessors are a set of users to whom the authorization is granted. Accessors can be represented with a set of user names, the friendship or a set of group names in OSNs. To facilitate collaborative privacy management, we further introduce *trust levels*, which are assigned to accessors when defining the privacy policies. Golbeck [14] discussed how trust could be used in OSNs, focusing on OSNs for collaborative rating. We believe that such considerations can also apply to our privacy management scenario. As addressed in Section 3.2.2, trust is one of the factors in our approach for resolving privacy conflicts. Clearly, in our scenario, trust has a different meaning from the one used in [14]. The notation of trust in our work mainly convey information about how much confidence a controller put on her/his friends who would not disclose the sensitive information to untrusted users. Also, trust levels can be changed in different situations. The notion of accessor specification is formally defined as follows:

DEFINITION 6. *(Accessor Specification). Let ac be a user $u \in U$, the friendship,[1] or a group $g \in G$, that is, $ac \in U \cup \{friendOf\} \cup G$. Let tl be a trust level, which is a rational number in the range [0,1], assigned to ac. And let $at \in \{UN, FS, GN\}$ be the type of the accessor (*user name, friendship, *and* group name, *respectively). The accessor specification is defined as a set, $\{a_1, \ldots, a_n\}$, where each element is a 3-tuple $< ac, tl, at >$.*

Data Specification: In the context of OSNs, user data is composed of three types of information. *User profile* describes who a user is in the OSN, including identity and personal information, such as name, birthday, interests and contact information. *User friendship* shows who a user knows in the OSN, including a list of friends to represent connections with family, coworkers, colleagues, and so on. *User content* indicates what a user has in the OSN, including photos, videos, statues, and all other data objects created through various activities in the OSN.

Again, to facilitate effective resolution of privacy conflicts for collaborative privacy control, we introduce *sensitivity levels* for data specification, which are assigned by the controllers to the shared data. The users' judgment of the sensitivity levels of the data is not binary (private/public), but multi-dimensional with varying degrees of sensitivity. Formally, the data specification is defined as follows:

DEFINITION 7. *(Data Specification). Let $d \in D$ be a data item, and sl be a sensitivity level, which is a rational number in the range [0,1], assigned to d. The data specification is defined as a tuple $< d, sl >$.*

Privacy Policy: To summarize the aforementioned features and elements, we introduce a formal definition of privacy policies for collaborative data sharing as follows:

DEFINITION 8. *(Privacy Policy). A privacy policy is a 4-tuple $P =< controller, accessor, data, effect >$, where*

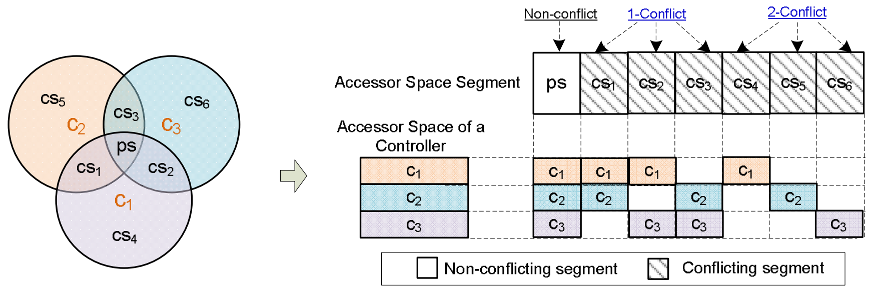
- *$controller$ is a controller specification defined in Definition 5;*
- *$accessor$ is an access specification defined in Definition 6;*
- *$data$ is a data specification defined in Definition 7; and*
- *$effect \in \{permit, deny\}$ is the authorization effect of the policy.*

Suppose the trust levels that a controller can allocate to a user or a user set are {0.00, 0.25, 0.50, 0.75, 1.00}, indicating *none* trust, *weak* trust, *medium* trust, *strong* trust, and *strongest* trust, respectively. Similarly, a controller can leverage five sensitivity levels: 0.00 (*none*), 0.25 (*low*), 0.50 (*medium*), 0.75 (*high*), and 1.00 (*highest*) for the shared data. The following is an example of privacy policy in terms of our policy specification scheme.

EXAMPLE 1. *$Alice$ authorizes users who are her friends or users in hiking group to access a photo (identified by a particular* photoId*) she is tagged in, where $Alice$ considers her friends with a* medium *trust level, the hiking group with a* weak *trust level, and the photo with a* high *sensitivity level:*

$p = (< Alice, ST >, \{< friendOf, 0.5, FS >, < hiking, 0.25, GN >\}, < photoId, 0.75 >, permit)$.

[1]We limit our consideration to $friendOf$ relation. The support of more relations such as $colleagueOf$ and $classmateOf$ does not significantly complicate our approach proposed in this paper.



(a) Accessor space segmentation (b) Privacy conflict identification

Figure 2: Example of Privacy Conflict Identification Based on Accessor Space Segmentation.

3.2 Identifying and Resolving Privacy Conflicts

When two users disagree on whom the shared data item should be exposed to, we say a *privacy conflict* occurs. The essential reason leading to the privacy conflicts is that multiple associated users of the shared data item often have different privacy concerns over the data item. For example, assume that Alice and Bob are two controllers of a photo. Each of them defines a privacy policy stating only her/his friends can view this photo. Since it is almost impossible that Alice and Bob have the same set of friends, privacy conflicts may *always* exist considering collaborative control over the shared data item.

A *naive* solution for resolving multiparty privacy conflicts is to only allow the *common* users of accessor sets defined by the multiple controllers to access the data [24]. Unfortunately, this solution is too restrictive in many cases and may not produce desirable results for resolving multiparty privacy conflicts. Let's consider an example that four users, Alice, Bob, Carol and Dave, are the controllers of a photo, and each of them allows her/his friends to see the photo. Suppose that Alice, Bob and Carol are close friends and have many common friends, but Dave has no common friends with them and has a pretty weak privacy concern on the photo. In this case, adopting the *naive* solution for conflict resolution may turn out that no one can access this photo. Nevertheless, it is reasonable to give the view permission to the common friends of Alice, Bob and Carol. A *strong* conflict resolution strategy may provide a better privacy protection. Meanwhile, it may reduce the social value of data sharing in OSNs. Therefore, it is important to consider the tradeoff between *privacy protection* and *data sharing* when resolving privacy conflicts. To address this issue, we introduce a mechanism for identifying multiparty privacy conflicts, as well as a systematic solution for resolving multiparty privacy conflicts.

3.2.1 *Privacy Conflict Identification*

Through specifying the privacy policies to reflect the privacy concern, each controller of the shared data item defines a set of trusted users who can access the data item. The set of trusted users represents an *accessor space* for the controller. In this section, we first introduce a space segmentation approach [16] to partition accessor spaces of all controllers of a shared data item into disjoint segments. Then, conflicting accessor space segments (called *conflicting segments* in the rest of this paper), which contain accessors that some controllers of the shared data item do not trust, are identified. Each conflicting segment contains at least one privacy conflict.

Algorithm 1 shows the pseudocode of generating conflicting accessor space segments for all controllers of a shared data item. An entire accessor space derived from the policies of all controllers of shared data item is first partitioned into a set of disjoint segments.

Algorithm 1: Identification of Conflicting Accessor Space

```
   Input: A set of accessor space, $A$.
   Output: A set of disjoined conflicting accessor spaces, $CS$.
 1 /* Partition the entire accessor space */
 2 $S \longleftarrow Partition(A)$;
 3 /* Identify the conflicting segments */
 4 $CS.New()$;
 5 foreach $s \in S$ do
 6     /* Get all controllers associated with a segment s */
 7     $C \longleftarrow GetControllers(s)$;
 8     if $|C| < |A|$ then
 9         $CS.Append(s)$;

10 Partition($A$)
11 foreach $a \in A$ do
12     $s_a \longleftarrow FriendSet(a)$;
13     foreach $s \in S$ do
14         /* $s_a$ is a subset of s*/
15         if $s_a \subset s$ then
16             $S.Append(s \setminus s_a)$;
17             $s \longleftarrow s_a$;
18             $Break$;
19         /* $s_a$ is a superset of s*/
20         else if $s_a \supset s$ then
21             $s_a \longleftarrow s_a \setminus s$;
22         /* $s_a$ partially matches s*/
23         else if $s_a \cap s \neq \emptyset$ then
24             $S.Append(s \setminus s_a)$;
25             $s \longleftarrow s_a \cap s$;
26             $s_a \longleftarrow s_a \setminus s$;
27     $S.Append(s_a)$;
28 return $S$;
```

As shown in lines 10-28 in Algorithm 1, a function called `Partition()` accomplishes this procedure. This function works by adding an accessor space s_a derived from policies of an controller a to an accessor space set S. A pair of accessor spaces must satisfy one of the following relations: *subset* (line 14), *superset* (line 19), *partial match* (line 22), or *disjoint* (line 27). Therefore, one can utilize set operations to separate the overlapped spaces into disjoint spaces.

Conflicting segments are identified as shown in lines 5-9 in Algorithm 1. A set of conflicting segments $CS : \{cs_1, cs_2, \ldots, cs_n\}$ from the policies of conflicting controllers has the following three properties:

1. All conflicting segments are pairwise disjoint: $cs_i \cap cs_j = \emptyset, 1 \leq i \neq j \leq n$;
2. Any two different accessors a and $a^{'}$ within a single conflicting segment (cs_i) are defined by the exact same set of controllers: $GetController(a) = GetController(a^{'})$, where

$a \in cs_i, a^{'} \in cs_i, a \neq a^{'}$; [2] and

3. The accessors in any conflicting segments are untrusted by at least one controller of the shared data item.

Figure 2 gives an example of identifying privacy conflicts based on accessor space segmentation. We use circles to represent accessor spaces of three controllers, c_1, c_2 and c_3, of a shared data item. We can notice that three of accessor spaces overlap with each other, indicating that some accessors within the overlapping spaces are trusted by multiple controllers. After performing the space segmentation, seven disjoint accessor space segments are generated as shown in Figure 2 (a). To represent privacy conflicts in an intuitive way, we additionally introduce a grid representation of privacy conflicts, in which space segments are displayed along the horizontal axis of a matrix, controllers are shown along the vertical axis of the matrix, and the intersection of a segment and a controller is a grid that displays the accessor subspace covered by the segment. We classify the accessor space segments as two categories: *non-conflicting* segment and *conflicting* segment. *Non-conflicting* segment covers all controllers' access spaces, which means any accessor within the segment is trusted by all controllers of the shared data item, indicating no privacy conflict occurs. A *conflicting* segment does not contain all controllers' access spaces that means accessors in the segment are untrusted by some controllers. Each *untrusting* controller points out a privacy conflict. Figure 2 (b) shows a grid representation of privacy conflicts for the example. We can easily identify that the segment *ps* is a *non-conflicting* segment, and cs_1 through cs_6 are *conflicting* segments, where cs_1, cs_2 and cs_3 indicate *one* privacy conflict, respectively, and cs_4, cs_5 and cs_6 are associated with *two* privacy conflicts, respectively.

3.2.2 Privacy Conflict Resolution

The process of privacy conflict resolution makes a decision to allow or deny the accessors within the conflicting segments to access the shared data item. In general, allowing the assessors contained in conflicting segments to access the data item may cause *privacy risk*, but denying a set of accessors in conflicting segments to access the data item may result in *sharing loss*. Our privacy conflict resolution approach attempts to find an optimal tradeoff between privacy protection and data sharing.

Measuring Privacy Risk: The privacy risk of a conflicting segment is an indicator of potential threat to the privacy of controllers in terms of the shared data item: the higher the privacy risk of a conflicting segment, the higher the threat to controllers' privacy. Our basic premises for the measurement of privacy risk for a conflicting segment are the following: (a) the lower the number of controllers who trust the accessors within the conflicting segment, the higher the privacy risk; (b) the stronger the general privacy concerns of controllers, the higher the privacy risk; (c) the more sensitive the shared data item, the higher the privacy risk; (d) the wider the data item spreads, the higher the privacy risk; and (e) the lower the trust levels of accessors in the conflicting segment, the higher the privacy risk. Therefore, the privacy risk of a conflicting segment is calculated by a monotonically increasing function with the following parameters:

- **Number of privacy conflicts**: The number of privacy conflicts in a conflicting segment is indicated by the number of the untrusting controllers. The untrusting controllers of a conflict segment i are returned by a function $controllers_{ut}(i)$;
- **General privacy concern of an untrusting controller**: The general privacy concern of an untrusting controller j is denoted as pc_j. The general privacy concern of a controller can be derived from her/his *default* privacy setting for data sharing. Different controllers may have different general privacy concern with respect to the same kinds of data. For example, public figures may have higher privacy concern on their shared photos than ordinary people;
- **Sensitivity of the data item**: Data sensitivity in a way defines controllers' perceptions of the confidentiality of the data being transmitted. The sensitivity level of the shared data item explicitly chosen by an untrusting controller j is denoted as sl_j. The factor depends on the untrusting controllers themself. Some untrusting controllers may consider the shared data item with the higher sensitivity;
- **Visibility of the data item**: The visibility of the data item with respect to a conflicting segment captures how many accessors are contained in the segment. The more the accessors in the segment, the higher the visibility; and
- **Trust of an accessor**: The trust level of an accessor k is denoted as tl_k, which is an average value of the trust levels defined by the trusting controllers of the conflicting segment for the accessor.

The privacy risk of a conflict segment i due to an untrusting controller j, denoted as $PR(i, j)$, is defined as

$$PR(i,j) = pc_j \otimes sl_j \otimes \sum_{k \in accessors(i)} (1 - tl_k) \tag{1}$$

where, function $accessors(i)$ returns all accessors in a segment i, and operator $\otimes$ is used to represent any arbitrary combination functions. For simplicity, we utilize the product operator.

In order to measure the *overall* privacy risk of a conflicting segment i, denoted as PR(i), we can use following equation to aggregate the privacy risks of i due to different untrusting controllers. Note that we can also use any combination function to combine the per-controller privacy risk. For simplicity, we employ the summation operator here.

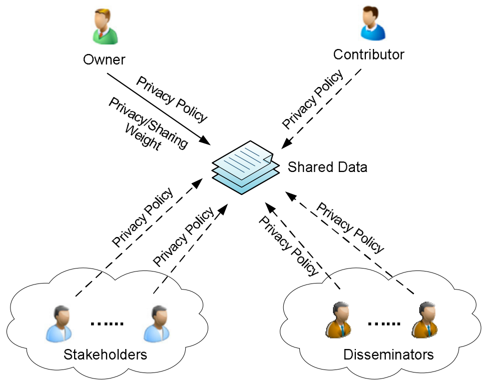
$$\begin{aligned} PR(i) &= \sum_{j \in controllers_{ut}(i)} (PR(i,j)) \\ &= \sum_{j \in controllers_{ut}(i)} (pc_j \times sl_j \times \sum_{k \in accessors(i)} (1 - tl_k)) \end{aligned} \tag{2}$$

Measuring Sharing Loss: When the decision of privacy conflict resolution for a conflicting segment is "deny", it may cause losses in potential data sharing, since there are controllers expecting to allow the accessors in the conflicting segment to access the data item. Similar to the measurement of the privacy risk, five factors are adopted to measure the sharing loss for a conflicting segment. Compared with the factors used for quantifying the privacy risk, the only difference is that we will utilize a factor, *number of trusting controllers*, to replace the factor, *number of privacy conflicts (untrusting controllers)*, for evaluating the sharing loss of a conflicting segment. The overall sharing loss $SL(i)$ of a conflicting segment i is computed as follows:

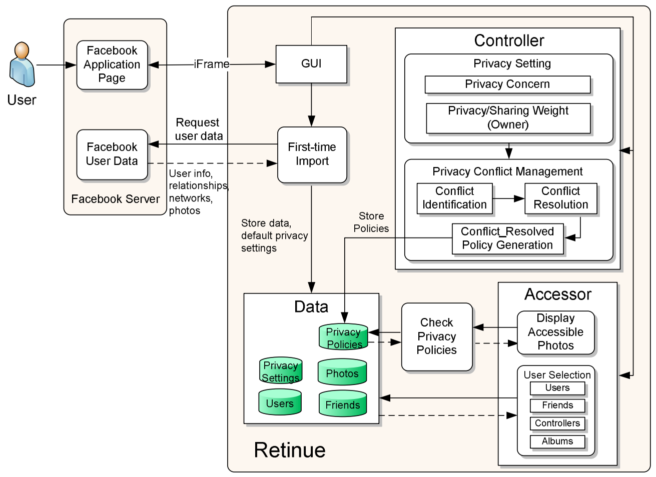
$$SL(i) = \sum_{j \in controllers_t(i)} ((1 - pc_j \times sl_j) \times \sum_{k \in accessors(i)} tl_k) \tag{3}$$

where, function $controllers_t(i)$ returns all trusting controllers of a segment i.

[2] $GetController()$ is a function that returns all controllers whose accessor spaces contain a specific accessor.



(a) Collaborative Control Overview



(b) Operational Components in *Retinue* Application

Figure 3: System Architecture of *Retinue*.

Privacy Conflict Resolution on the Tradeoff between Privacy Protection and Data Sharing: The tradeoff between privacy and utility in data publishing has been recently studied [8, 19]. Inspired by those work, we introduce a mechanism to balance privacy protection and data sharing for an effective privacy conflict resolution in OSNs.

An optimal solution for privacy conflict resolution should cause a little more privacy risk when allowing the accessors in some conflicting segments to access the data item, and gets lesser loss in data sharing when denying the accessors to access the shared data item. Thus, for each conflict resolution solution s, a resolving score $RS(s)$ can be calculated using the following equation:

$$RS(s) = \frac{1}{\alpha \sum_{i_1 \in CS_p^s} PR(i_1) + \beta \sum_{i_2 \in CS_d^s} SL(i_2)} \tag{4}$$

where, CS_p^s and CS_d^s denote *permitted* conflicting segments and *denied* conflicting segments respectively in the conflict resolution solution s. And α and β are preference weights for the privacy risk and the sharing loss, $0 \leq \alpha, \beta \leq 1$ and $\alpha + \beta = 1$.

Then, the optimal conflict resolution CR_{opt} on the tradeoff between privacy risk and sharing loss can be identified by finding the maximum resolving score:

$$CR_{opt} = \max_{s} RS(s) \tag{5}$$

To find the maximum resolving score, we can first calculate the privacy risk ($PR(i)$) and the sharing loss ($SL(i)$) for each conflict segment (i), individually. Finally, following equation can be utilized to make the decisions (permitting or denying conflicting segments) for privacy conflict resolution, guaranteeing to always find an optimal solution.

$$Decision = \begin{cases} \text{Permit} & \text{if } \alpha SL(i) \geq \beta PR(i) \\ \text{Deny} & \text{if } \alpha SL(i) < \beta PR(i) \end{cases} \tag{6}$$

3.2.3 *Generating Conflict-Resolved Policy*

Once the privacy conflicts are resolved, we can aggregate accessors in *permitted* conflicting segments CS_p and accessors in the *non-conflicting* segment ps (in which accessors should be always allowed to access the shared data item) together to generate a new accessor list (AL) as follows:

$$AL = (\bigcup_{i \in CS_p} Accessors(i)) \cup Accessors(ps) \tag{7}$$

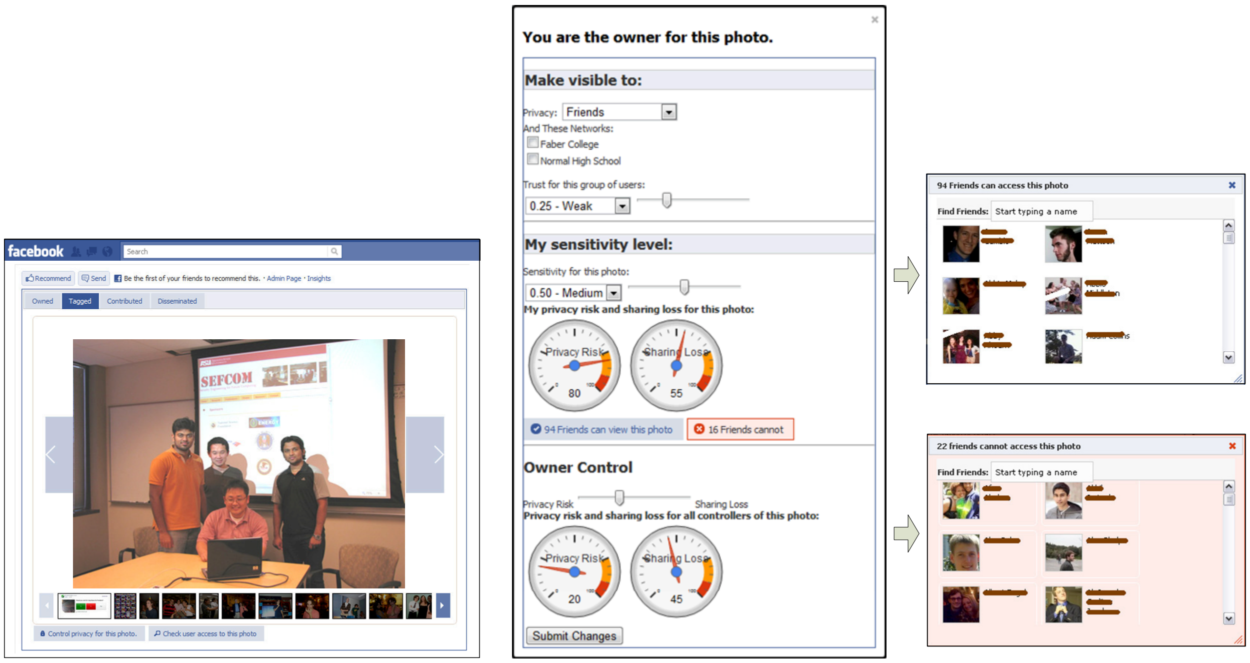
Using the example shown in Figure 2, we assume that cs_1 and cs_3 become *permitted* conflicting segments after resolving the privacy conflicts. Therefore, the aggregated accessor list can be derived as $AL = Accessors(cs_1) \cup Accessors(cs_3) \cup Accessors(ps)$. Finally, the aggregated accessor list is used to construct a conflict-resolved privacy policy for the shared data item. The generated policy will be leveraged to evaluate all access requests toward the data item.

4. IMPLEMENTATION AND EVALUATION

4.1 Prototype Implementation

We implemented a proof-of-concept Facebook application for the collaborative management of shared data called *Retinue* (http://apps.facebook.com/retinue_tool). Our prototype application enables multiple associated users to specify their privacy concerns to co-control a shared data item. *Retinue* is designed as a third-party Facebook application which is hosted in an Apache Tomcat application server supporting PHP and MySQL databases, with a user interface built using jQuery and jQuery UI and built on an AJAX-based interaction model. *Retinue* application is based on the iFrame external application approach. Using the Javascript and PHP SDK, it accesses users' Facebook data through the Graph API and Facebook Query Language. It is worth noting that our current implementation was restricted to handle photo sharing in OSNs. Obversely, our approach can be generalized to deal with other kinds of data sharing (e.g. videos and comments) in OSNs as long as the stakeholder of shared data are identified with effective methods like tagging or searching.

Figure 3 shows the system architecture of *Retinue*. The overview of collaborative control process is depicted in Figure 3(a), where the *owner* can regulate the access of the shared data. In addition, other controllers, such as *the contributor*, *stakeholders* and *disseminators*, can specify their privacy concerns over the shared data as well. To effectively resolve privacy conflicts caused by different privacy concerns of multiple controllers, the data *owner* can also adjust the preference weights for the privacy risk and the sharing


You are the owner for this photo.

Make visible to:

Privacy: Friends

And These Networks:

Faber College

Normal High School

Trust for this group of users:

0.25 - Weak

My sensitivity level:

Sensitivity for this photo:

0.50 - Medium

My privacy risk and sharing loss for this photo:

Privacy Risk 80 · Sharing Loss 55

94 Friends can view this photo · 16 Friends cannot

Owner Control

Privacy Risk · Sharing Loss

Privacy risk and sharing loss for all controllers of this photo:

Privacy Risk 20 · Sharing Loss 45

Submit Changes

94 Friends can access this photo

Find Friends: Start typing a name

[redacted]

22 friends cannot access this photo

Find Friends: Start typing a name

[redacted]

facebook
Search
Recommend
Send
Be the first of your friends to recommend this. · Admin Page · Insights
Owned
Tagged
Contributed
Disseminated
SEFCOM
Control privacy for this photo.
Check user access to this photo

(a) Main Interface.

(b) Controllers' Interfaces.

Figure 4: *Retinue* Interfaces.

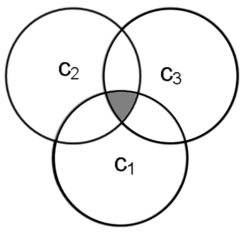
loss to make an appropriate privacy-sharing tradeoff. Figure 3(b) shows the core components in *Retinue* application and their interactions. The *Retinue* application is hosted on an external website, but is accessed on a Facebook application frame via an iFrame. The Facebook server handles login and authentication for the application, and other user data is imported on the user's first login. At this point, users are asked to specify their initial privacy settings and concerns for each type of photo. All photos are then imported and saved using these initial privacy settings. Users' networks and friend lists are imported from Facebook server as well. Once information is imported, a user accesses *Retinue* through the application page on Facebook, where s/he can query access information, complete privacy setting for photos in which s/he is a controller, and view photos s/he is allowed to access. The component for privacy conflict management in *Retinue* application is responsible for the privacy conflict detection and resolution, and the generation of conflict-resolved privacy policy, which is then used to evaluate access requests for the shared data.

A snapshot of the main interface of *Retinue* is shown in Figure 4 (a). All photos are loaded into a gallery-style interface. To access photos, a user clicks the "Access" tab and then s/he can view her/his friends' photos that s/he was authorized. To control photo sharing, a user clicks the "Owned", "Tagged", "Contributed", or "Disseminated" tabs, then selects any photo in the gallery to define her/his privacy preferences for that photo. The controllers' interfaces are depicted in Figure 4 (b). A controller can select the trusted groups of accessors and assign corresponding trust levels, as well as choose the sensitivity level for the photo. Also, the privacy risk and sharing loss for the controller with respect with the photo are displayed in the interface. In addition, the controller can immediately see how many friends can or cannot access the photo in the interface. If the controller clicks the buttons, which show the numbers of accessible or unaccessible friends, a window appears showing the details of all friends who can or cannot view the photo. The purpose of such feedback information is not only to give a controller the information of how many friends can or cannot access the photo, but as a way to react to results. If the controller is not satisfied with the current situation of privacy control, s/he may adjust her/his privacy settings, contact the owner of the photo to ask her/him to change the weights for the privacy risk and the sharing loss, or even report a privacy violation to request OSN administrators to delete the photo. If the user is the owner of the photo, s/he can also view the overall privacy risk and sharing loss for the shared photo, and has the ability to adjust the weights to balance privacy protection and data sharing of the shared photo.

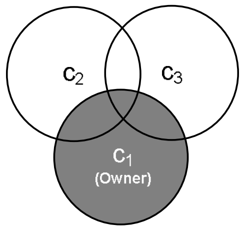
4.2 Evaluation and Experiments

4.2.1 Evaluation of Privacy Conflict Resolution

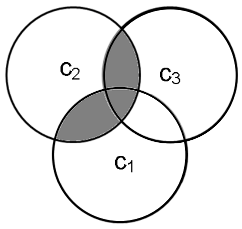
We evaluate our approach for privacy conflict resolution by comparing our solution with the *naive solution* and the privacy control solution used by existing OSNs, such as Facebook (simply called *Facebook solution* in the rest of this paper) with respect to two metrics, *privacy risk* and *sharing loss*. Consider the example demonstrated in Figure 2, where three controllers desire to regulate access of a shared data item. The *naive solution* is that only the accessors in the non-conflicting segment are allowed to access the data item as shown in Figure 5(a). Thus, the *privacy risk* is always equal to 0 for this solution. However, the *sharing loss* is the absolute maximum, as all conflicting segments, which may be allowed by at least one controller, are always denied. The *Facebook solution* is that the owner's decision has the highest priority. All accessors within the segments covered by the owner's space are allowed to access the data item, but all other accessors are denied as illustrated in Figure 5(b). This is, obviously, ideal for the owner, since her/his *privacy risk* and *sharing loss* are both equal to 0. However, the *privacy risk* and the *sharing loss* are large for every non-owner controller.



(a) Naive Solution.

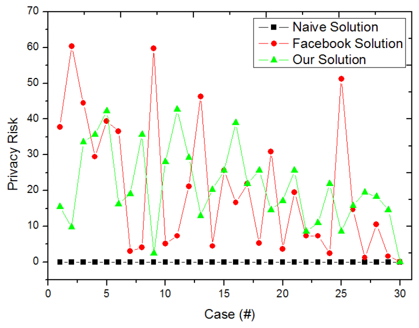


(b) Facebook Solution.

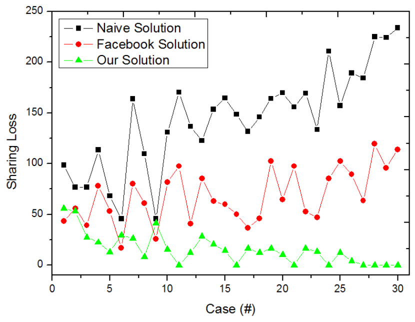


(c) Our Solution.

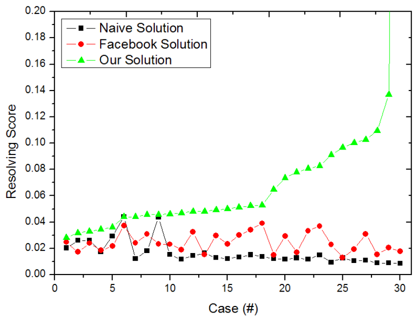
Figure 5: Example of Resolving Privacy Conflicts.



(a) Privacy Risk.



(b) Sharing Loss.



(c) Resolving Score.

Figure 6: Conflict Resolution Evaluation.

For our solution, each conflicting segment is evaluated individually. Using the same example given in Figure 2, suppose cs_1 and cs_3 become *permitted* conflicting segments after resolving the privacy conflicts. Figure 5(c) demonstrates the result of our privacy conflict resolution. Our solution make a tradeoff between privacy protection and data sharing by maximizing the resolving score, which is a combination of *privacy risk* and *sharing loss*. The worst case of our solution is the same as the *naive solution*–only mutually permitted accessors are allowed to access the data item. However, this case only occurs when strong privacy concerns are indicated by each controller. On the other hand, if all accessors have pretty weak privacy concerns, all accessors in conflicting segments may be allowed to access the data, which is not possible with either of other two solutions. Such a case leads to a sharing loss of 0, but does not have an significantly increased privacy risk against other two solutions.

To quantitatively evaluate our solution, our experiment used cases where there are three controllers of shared data items and assume that each controller has indicated to allow her/his friends to view the data item. We also utilized the average number of user friends, 130, which is claimed by Facebook statistics [3]. Additionally, we assume all controllers share 30 friends with each other, 10 of which are shared among everyone (common users). All settings including privacy concerns, sensitivity levels, and trust levels were randomized for each case, and the privacy risk, sharing loss, and resolving score for each case were calculated. To represent the data sensibly, we sorted the samples from lowest resolving score to highest under our evaluation. Figure 6 shows our experimental results with respect to randomly generated 30 user cases.

In Figure 6(a), we can observe that the privacy risks for the *naive solution* are always equal to 0, since no untrusted accessors are allowed to view the data item. The privacy risks for *Facebook solution* and our solution wavered. Obviously, this depends greatly on the settings of the non-owner controllers. If these controllers are apathetic toward the shared data item, *Facebook solution* will be preferable. However, it should be noted that *Facebook solution* had very high extrema. This is avoided in our solution where high privacy risks will usually result in denying access.

Unsurprisingly, the sharing loss for the *naive solution* was always the highest, and often higher than both other two solutions as shown in Figure 6(b). Our solution usually had the lowest sharing loss, and sometimes is equivalent to the *naive* or *Facebook solution*, but rarely greater than. One may notice that the sharing loss is very low compared to the privacy risks in our experience. This is an inherent effect of our solution itself–if sharing loss is very high, users will be granted access to the data item, changing this segment's sharing loss to zero.

As we can notice from Figure 6(c), the resolving score for our solution is always as good as or better than the *naive* or *Facebook solution*. In our sample data, it was usually significantly better, and rarely was the same as either of other two solutions. It further indicates that our solution can always achieve a good tradeoff between privacy protection and data sharing for privacy conflict resolution.

4.2.2 Evaluation of System Usability

Participants and Procedure: *Retinue* is a functional proof-of-concept implementation of collaborative privacy management. To measure the practicality and usability of our mechanism, we conducted a survey study (n=30) to explore the factors surrounding users' desires for privacy controls such as those implemented in *Retinue*. Particularly, we were interested in users' perspectives on the current Facebook privacy system and their desires for more control over photos they do not own. We recruited participants through university mailing lists and through Facebook itself using Facebook's built-in sharing API. Users were given the opportunity to share our application and play with their friends. While this is not a ran-

Table 1: Usability Evaluation for Facebook and *Retinue* Privacy Controls.

Metric		Facebook		Retinue	
		Average	Upper bound on 95% confidence interval	Average	Lower bound on 95% confidence interval
Likability		0.39	0.44	0.74	0.72
Understanding		0.33	0.36	0.69	0.65
Control	Sharing with Trusted Users	0.36	0.40	0.69	0.66
	Protecting from Untrusted Users	0.30	0.35	0.71	0.70

dom sampling, recruiting using the natural dissemination features of Facebook arguably gives an accurate profile of the ecosystem.

In our user study (http://bit.ly/retinue_study), participants were asked to first answer some questions about their usage and perception of Facebook's privacy controls. Users were then instructed to install the application using their Facebook profiles and complete the following actions: set privacy settings for a photo they do not own, set privacy settings for a photo they own, and answer questions about their understanding. As users completed these actions, they were asked questions on the usability of the controls in Retinue.

User Study of Retinue: The criteria for usability evaluation were split into three areas: *likeability*, *understanding*, and *control*. *Likeability* is simply a measure of a user's basic opinion of a particular feature or control. While this does not provide specific feedback for improvement, it can help identify what aspects of sharing and control are important to a user. *Understanding* is a measure of how intuitive the concepts and controls are. This is tremendously useful for improving the usability of controls. *Control* is a measure of the user's perceived control of their personal data. *Control*, in addition, can be narrowed down into the areas of *sharing with trusted users* and *protecting from untrusted users*. While this is not a definitive measure of privacy, making a user feel safe is almost as important as protecting a user. Questions were measured on a three- or four-point scale (scaled from 0 to 1 for numerical analysis). For measurement analysis, a higher number is used to indicate a positive opinion or perception, while a lower number is used to indicate a negative one. We were interested in the average user perception of the system, so we analyzed a 95% confidence interval for the users' answers. This assumes the population to be mostly normal.

Before using *Retinue*, users were asked a few questions about their usage of Facebook to determine the user's perceived usability of the current Facebook's privacy controls. This included questions on *likeability* (e.g. "indicate how much you like privacy features for photos you are tagged in"), *understanding* (e.g. "indicate how much you understand how to prevent certain people from seeing photos I am tagged in"), and *control* (e.g. "indicate how in control you feel when sharing photos I own with people I want to"). For our confidence interval, we were interested in determining the average user's maximum positive opinion of Facebook's privacy controls, so we looked at the upper bound of the confidence interval.

An average user asserts at most 44% positively about the *likability*, 40% positively about *sharing control*, 35% positively about *protection control* and 36% on their *understanding* of Facebook's privacy mechanisms as shown in Table 1. This demonstrates an average negative opinion of the Facebook's privacy controls that users currently must use.

After Using *Retinue*, users were then asked to perform a few tasks in *Retinue* and were asked a few questions to determine the users perceived usability of *Retinue*. This also included questions on *likeability* (e.g. "indicate how much you like the *trust level* feature"), *understanding* (e.g. "indicate your understanding of the meaning of *sharing loss*"), and *control* (e.g. "please indicate how in control you feel when sharing photos I own with the people I want to"). For our confidence interval, we were interested in determining the average user's minimum positive opinion of *Retinue*'s privacy controls, so we looked at the lower bound of the confidence interval.

An average user asserts at least 72% positively on *likeability*, 65% positively on *understanding*, 66% on *sharing control* and 70% on *protection control* as shown in Table 1. This demonstrates an average positive opinion of the controls and ideas presented to users in *Retinue*.

5. RELATED WORK

Several proposals of an access control scheme for OSNs have been introduced (e.g., [9, 10, 12, 13, 17]). Carminati et al. [9] introduced a trust-based access control mechanism, which allows the specification of access rules for online resources where authorized users are denoted in terms of the relationship type, depth, and trust level between users in OSNs. They further presented a semi-decentralized discretionary access control system and a related enforcement mechanism for controlled sharing of information in OSNs [10]. Fong et al. [13] proposed an access control model that formalizes and generalizes the access control mechanism implemented in Facebook. Gates [11] described relationship-based access control as one of the new security paradigms that addresses the requirements of the Web 2.0. Then, Fong [12] recently formulated this paradigm called a Relationship-Based Access Control (ReBAC) that bases authorization decisions on the relationships between the resource owner and the resource accessor in an OSN. However, none of these work could accommodate privacy control requirements with respect to the *collaborative* data sharing in OSNs.

Several recent work [7, 15, 18, 22, 24] recognized the need of joint management for data sharing, especially photo sharing, in OSNs. In particular, Squicciarini et al. [22] proposed a solution for collective privacy management for photo sharing in OSNs. This work considered the privacy control of a content that is co-owned by multiple users in an OSN, such that each co-owner may separately specify her/his own privacy preference for the shared content. The Clarke-Tax mechanism was adopted to enable the collective enforcement for shared content. Game theory was applied to evaluate the scheme. However, a general drawback of this solution is the usability issue, as it could be very hard for ordinary OSN users to comprehend the Clarke-Tax mechanism and specify appropriate bid values for auctions. In addition, the auction process adopted in their approach indicates only the winning bids could determine who was able to access the data, instead of accommodating all stakeholders' privacy preferences. In contrast, our work proposes a simple but flexible mechanism for collaborative management of shared data in OSNs. In particular, we introduce an effective conflict resolution solution, which makes a tradeoff between privacy protection and data sharing considering the privacy concerns from multiple associated users.

Measuring privacy risk in OSNs has been addressed recently by several work [6, 20, 23]. Becker et al. [6] presented *PrivAware*, a tool to detect and report unintended information loss through quantifying privacy risk associated with friend relationship in OSNs. In [23], Talukder et al. discussed a privacy protection tool, called *Privometer*, which can measure the risk of potential privacy leakage cased by malicious applications installed in the user's friend profiles and suggest self-sanitization actions to lessen this leakage accordingly. Liu et al. [20] proposed a framework to compute the privacy score of a user, indicating the user's potential risk caused by her/his participation in OSNs. Their solution also focused on the privacy settings of users with respect to their profile items. Compared with those existing work, our approach measures the privacy risk caused by different privacy concerns from multiple users, covering profile sharing, friendship sharing, as well as content sharing in OSNs.

6. CONCLUSION

In this paper, we have proposed a novel solution for privacy conflict detection and resolution for collaborative data sharing in OSNs. Our conflict resolution mechanism considers privacy-sharing tradeoff by quantifying privacy risk and sharing loss. Also, we have described a proof-of-concept implementation of our solution called *Retinue*, along with the extensive evaluation of our approach. As part of future work, we will formulate a comprehensive access control model to capture the essence of collaborative authorization requirements for data sharing in OSNs. Also, we would extend our work to address security and privacy challenges for emerging information sharing services such as location sharing [1] and other social network platforms such as Google+ [5].

Acknowledgments

This work was partially supported by the grants from National Science Foundation (NSF-IIS-0900970 and NSF-CNS-0831360) and Department of Energy (DE-SC0004308).

7. REFERENCES

[1] Facebook Places. `http://www.facebook.com/places/`.

[2] Facebook Privacy Policy. `http://www.facebook.com/policy.php/`.

[3] Facebook Statistics. `http://http://www.facebook.com/press/info.php?statistics`.

[4] Google+ Privacy Policy. `http://http://www.google.com/intl/en/+/policy/`.

[5] The Google+ Project. `https://plus.google.com`.

[6] J. Becker and H. Chen. Measuring privacy risk in online social networks. In *Proceedings of the 2009 Workshop on Web*, volume 2. Citeseer.

[7] A. Besmer and H. Richter Lipford. Moving beyond untagging: Photo privacy in a tagged world. In *Proceedings of the 28th international conference on Human factors in computing systems*, pages 1563–1572. ACM, 2010.

[8] J. Brickell and V. Shmatikov. The cost of privacy: destruction of data-mining utility in anonymized data publishing. In *Proceeding of the 14th ACM SIGKDD*, pages 70–78. ACM, 2008.

[9] B. Carminati, E. Ferrari, and A. Perego. Rule-based access control for social networks. In *On the Move to Meaningful Internet Systems 2006: OTM 2006 Workshops*, pages 1734–1744. Springer, 2006.

[10] B. Carminati, E. Ferrari, and A. Perego. Enforcing access control in web-based social networks. *ACM Transactions on Information and System Security (TISSEC)*, 13(1):1–38, 2009.

[11] E. Carrie. Access Control Requirements for Web 2.0 Security and Privacy. In *Proc. of Workshop on Web 2.0 Security & Privacy (W2SP)*. Citeseer, 2007.

[12] P. Fong. Relationship-Based Access Control: Protection Model and Policy Language. In *Proceedings of the First ACM Conference on Data and Application Security and Privacy*. ACM, 2011.

[13] P. Fong, M. Anwar, and Z. Zhao. A privacy preservation model for facebook-style social network systems. In *Proceedings of the 14th European conference on Research in computer security*, pages 303–320. Springer-Verlag, 2009.

[14] J. Golbeck. Computing and applying trust in web-based social networks. Ph.D. thesis, University of Maryland at College Park College Park, MD, USA. 2005.

[15] H. Hu and G. Ahn. Multiparty authorization framework for data sharing in online social networks. In *Proceedings of the 25th annual IFIP WG 11.3 conference on Data and applications security and privacy*, DBSec'11, pages 29–43. Springer, 2011.

[16] H. Hu, G. Ahn, and K. Kulkarni. Anomaly discovery and resolution in web access control policies. In *Proceedings of the 16th ACM symposium on Access control models and technologies*, pages 165–174. ACM, 2011.

[17] S. Kruk, S. Grzonkowski, A. Gzella, T. Woroniecki, and H. Choi. D-FOAF: Distributed identity management with access rights delegation. *The Semantic Web–ASWC 2006*, pages 140–154, 2006.

[18] A. Lampinen, V. Lehtinen, A. Lehmuskallio, and S. Tamminen. We're in it together: interpersonal management of disclosure in social network services. In *Proceedings of the 2011 annual conference on Human factors in computing systems*, pages 3217–3226. ACM, 2011.

[19] T. Li and N. Li. On the tradeoff between privacy and utility in data publishing. In *Proceedings of the 15th ACM SIGKDD*, pages 517–526. ACM, 2009.

[20] K. Liu and E. Terzi. A framework for computing the privacy scores of users in online social networks. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 5(1):6, 2010.

[21] M. Madejski, M. Johnson, and S. Bellovin. The Failure of Online Social Network Privacy Settings. Technical Report CUCS-010-11, Columbia University, NY, USA. 2011.

[22] A. Squicciarini, M. Shehab, and F. Paci. Collective privacy management in social networks. In *Proceedings of the 18th international conference on World wide web*, pages 521–530. ACM, 2009.

[23] N. Talukder, M. Ouzzani, A. Elmagarmid, H. Elmeleegy, and M. Yakout. Privometer: Privacy protection in social networks. In *Proceedings of 26th International Conference on Data Engineering Workshops (ICDEW)*, pages 266–269. IEEE, 2010.

[24] K. Thomas, C. Grier, and D. Nicol. unFriendly: Multi-party Privacy Risks in Social Networks. In *Privacy Enhancing Technologies*, pages 236–252. Springer, 2010.

[25] G. Wondracek, T. Holz, E. Kirda, and C. Kruegel. A practical attack to de-anonymize social network users. In *2010 IEEE Symposium on Security and Privacy*, pages 223–238. IEEE, 2010.

[26] E. Zheleva and L. Getoor. To join or not to join: the illusion of privacy in social networks with mixed public and private user profiles. In *Proceedings of the 18th international conference on World wide web*, pages 531–540. ACM, 2009.

Social Snapshots: Digital Forensics for Online Social Networks

Markus Huber*† Martin Mulazzani* Manuel Leithner*

Sebastian Schrittwieser* Gilbert Wondracek‡ Edgar Weippl*

SBA Research*
{mhuber, mmulazzani, mleithner, sschrittwieser, eweippl}@sba-research.org

Vienna PhD school of informatics† Vienna University of Technology‡

ABSTRACT

Recently, academia and law enforcement alike have shown a strong demand for data that is collected from online social networks. In this work, we present a novel method for harvesting such data from social networking websites. Our approach uses a hybrid system that is based on a custom add-on for social networks in combination with a web crawling component. The datasets that our tool collects contain profile information (user data, private messages, photos, etc.) and associated meta-data (internal timestamps and unique identifiers). These *social snapshots* are significant for security research and in the field of digital forensics. We implemented a prototype for Facebook and evaluated our system on a number of human volunteers. We show the feasibility and efficiency of our approach and its advantages in contrast to traditional techniques that rely on application-specific web crawling and parsing. Furthermore, we investigate different use-cases of our tool that include consensual application and the use of sniffed authentication cookies. Finally, we contribute to the research community by publishing our implementation as an open-source project.

Keywords

online social networks, forensics, security

1. INTRODUCTION

Over the past years, Online Social Networks (OSNs) have become the largest and fastest growing websites on the Internet. OSNs, such as Facebook or LinkedIn, contain sensitive and personal data of hundreds of millions of people, and are integrated into millions of other websites [11]. Research has acknowledged the importance of these websites and recently, a number of publications have focused on security issues that are associated with OSNs. In particular, a number of empirical studies on online social networks [1, 15, 18, 17, 29] highlight challenges to the security and privacy of social network users and their data.

We found that these, and similar studies, heavily depend on datasets that are collected from the social networking websites themselves, often involving data that is harvested from user profiles. Furthermore, as social networks continue to replace traditional means of digital storage, sharing, and communication, collecting this type of data is also fundamental to the area of digital forensics. For example, data from OSNs have been used successfully by criminal investigators to find criminals and even confirm alibis in criminal cases [7, 27].

While traditional digital forensics is based on the analysis of file systems, captured network traffic or log files, new approaches for extracting data from social networks or cloud services are needed. Interestingly and contrary to our intuition, we found little academic research that aims at developing and enhancing techniques for collecting this type of data efficiently. Despite the growing importance of data from OSNs for research, current state of the art methods for data extraction seem to be mainly based on custom web crawlers. However, we found this naïve approach to have a number of shortcomings:

- High network traffic: The extraction of profile data via traditional web crawling can be regarded as costly with regard to the required network resources, as it typically incurs a large amount of HTTP traffic and causes a high number of individual network connections. Apart from inherent disadvantages, social networking websites may also choose to block network access for clients that cause high levels of traffic, thus preventing them from harvesting additional data.

- Additional or hidden data: Per definition, web crawlers can only collect data that is accessible on the target website. However, we found that social networks often publish interesting meta-information (e.g. content creation timestamps or numeric identifiers) in other data sources, for example via developer APIs.

- Maintainability: The structure and layout of websites tend to change unpredictably over time. Additionally, the increasing use of dynamic or interpreted content (for example, JavaScript) leads to high maintenance requirements for custom web crawlers.

In this work, we introduce a novel method for data collection from social networks that aims to overcome these problems. Our approach is based on a hybrid system that uses an automated web browser in combination with an OSN third-party application. We show that our system can be used efficiently to gather "*social snapshots*", datasets which include user data and related information from the social network.

The main contributions of our work include:

- We introduce novel techniques to efficiently gather data from online social networks that may be used as criminal evidence. Our tool gathers more data than possible with today's approaches and it makes it feasible to link "online evidence" to traditional forensic artifacts found on computers using state-of-the-art tools (e.g. Encase).
- We implemented a prototype application that is aimed at Facebook, and released it under an open-source license.
- We performed an experimental evaluation involving a real-world test with volunteers and show results.

The rest of the paper is organized as follows: Section 2 introduces digital forensics followed by Section 3 which describes the design of our social snapshot framework. We evaluate the feasibility of social snapshots in Section 4. Section 5 discusses our results, Section 6 surveys related work and Section 7 concludes.

2. BACKGROUND

Digital forensics has received increasing attention in recent years as more and more crimes are commited exclusively or with the involvement of computers. Digital traces help courts and law enforcement agencies to capture valuable evidence for investigations. Existing research as well as applications in the area of digital forensics focus on filesystems [5], volatile memory [6, 16], databases [13], network traffic [8] and of course logfiles. The emergence of new online services replaces the traditional means of digital storage, sharing, and communication [4]. While traditional forensic approaches rely on the seizure of the suspect's hardware (computer, smartphone, storage media) for analysis, the emergence of online services, social networks and novel online communication methods can render this approach useless: A techno-savvy adversary might use a computer without hard disk, communicating securely with the use of encryption and storing files distributed all over the world. This would leave no traces locally for the forensic examiners to work with as soon as the computer is shut down. Another problem is the worldwide distribution of the Internet with its multitude of jurisdictions: while a court might order a company that is located within the same country to reveal information about a suspect, across borders this request may not stand.

With hundreds of millions of people sharing and communicating on social networks, they become more and more important for crime scene investigations. Traditional approaches to forensics on cloud computing and social network forensics are insufficient from an organizational as well as a technical point of view [2, 26]: the physical location of server systems is only known to the company, making seizure of hardware for examination in a forensic lab infeasible.

Often, the social network operator in question cooperates with law enforcement but in an equal number of cases they do not. Delicts that might happen solely within social networks such as cyber- stalking, mobbing or grooming, in combination with cross-border jurisdictions make it very hard to gather evidence in a forensically sound manner. A sample of social network related crimes can be found in [28]. With the increasing number of users we expect the number of social network related investigations to increase heavily in the near future. The Electronic Frontier Foundation (EFF) compiled a report [9] on U.S. law-enforcement agencies' access to social networking data. Most social networking providers have dedicated services to cater for law-enforcement requests. For example, Facebook offers two types of data: basic subscriber information ("Neoselect") and extended subscriber information ("Neoprint"). Our social snapshot application resembles a Neoprint whereas the entire subscriber content is fetched. Our social snapshot application thus offers an alternative for evidence collection, especially for non U.S. law-enforcement agencies.

3. DESIGN

Our digital forensics application enables an investigator to snapshot a given online social network account including meta-information, a method we termed "social snapshot". Meta-information such as exact timestamps are not available to the user via the user interface of the web application. A social snapshot represents the online social networking activity of a specific user such as circle of friends, exchanged messages, posted pictures etc. Due to the diversity of information available via OSNs we propose a twofold approach: an automated web-browser in combination with a custom third-party application. The social snapshot application is initialized with a user's credentials or authentication cookie. In the following, a custom third-party application is temporarily added to the target account. This application fetches the user's data, pictures, friend list, communication, and more. Information that is unavailable through the third-party application is finally gathered using traditional web-crawling techniques. By automating a standard web-browser and avoiding aggressive web-crawling we simulate the behavior of a human OSN user, thus minimizing the risk of being blocked by the social networking site. In this section, we describe the design of our approach as well as the individual components of our digital forensic framework.

3.1 Social Snapshot Framework

Figure 1 shows the core framework of our social snapshot application. **(1)** The social snapshot client is initialized by providing the target user's credentials or cookie. Our tool then starts the automated browser with the given authentication mechanism. **(2)** The automated browser adds our social snapshot application to the target user's profile and sends the shared API secret to our application server. **(3)** The social snapshot application responds with the target's contact list. **(4)** The automated web browser requests specific web pages of the user's profile and her contact list. **(5)** The received crawler data is parsed and stored. **(6)** While the automated browser requests specific web pages our so-

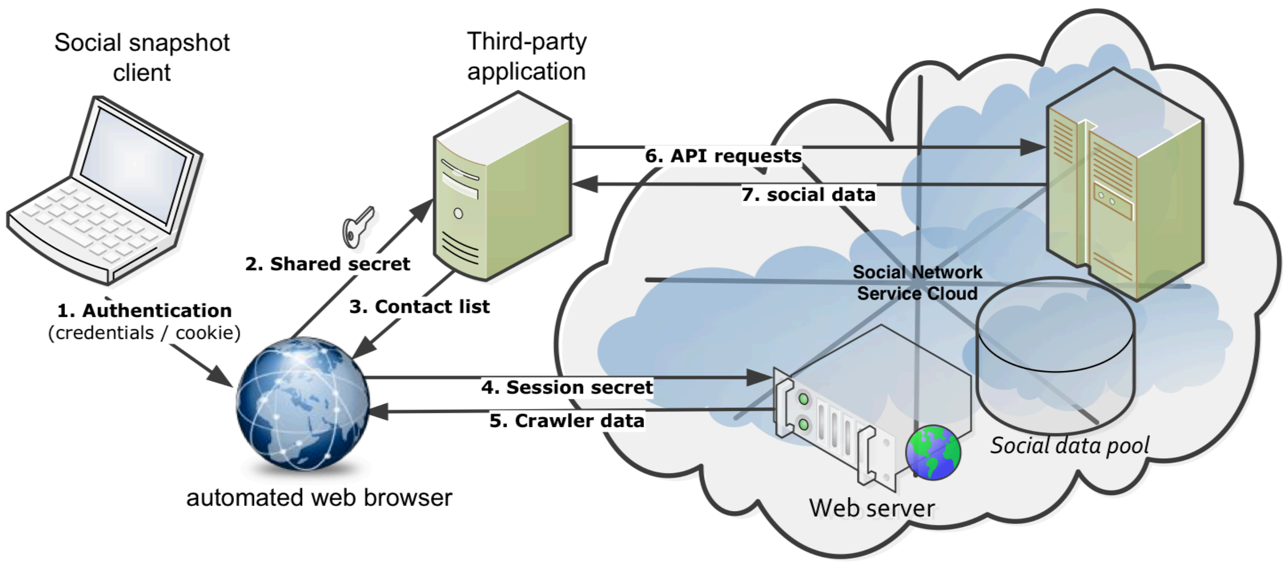


Figure 1: Collection of digital evidence through our social snapshot application.

cial snapshot application gathers personal information via the OSN API. **(7)** Finally the social data collected via the third-party application is stored on the social snapshot application server.

3.2 Authentication

In order to get access to the complete content of a target's social network account, social snapshots depend on gathering the initial authentication token. In the following, we outline three digital forensic scenarios that explain how this initial gathering of the authentication token works and that are representative for real-world use cases.

Consent. This naïve approach requires consent from the person whose social networking profiles are analyzed. A person would provide the forensic investigator temporary access to her social networking account in order to create a snapshot. This would also be the preferred method for academic studies to conduct this research in an ethically correct way and to comply with data privacy laws. We used this method for the evaluation of our proposed application as further described in Section 4.

Hijack social networking sessions. Our social snapshot application provides a module to hijack established social networking sessions. An investigator would monitor the target's network connection for valid authentication tokens, for example unencrypted WiFi connections or LANs. Once the hijack module finds a valid authentication token, the social snapshot application spawns a separate session to snapshot the target user's account.

Extraction from forensic image. Finally, physical access to the target's personal computer could be used to extract valid authentication cookies from web-browsers. Stored authentication cookies can be automatically found searching a gathered hard drive image or live analysis techniques such as Forenscope [6].

3.3 Depth of Information Collection

Starting from a target profile, a number of subsequent

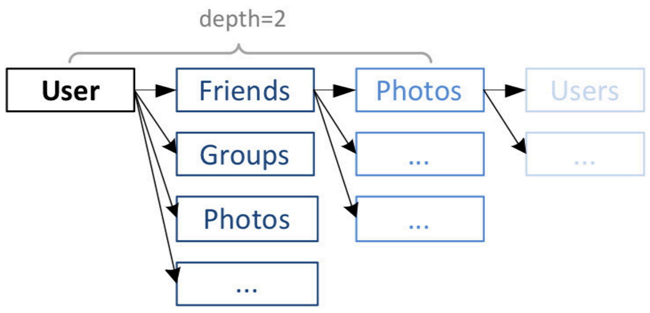


Figure 2: Example for elements fetched with social snapshot of depth=2

elements become available for crawling such as the user's friends, uploaded photos and joined groups. With these elements, again, a number of subsequent elements can be accessed. For example, the single-view page of a photo can contain comments and likes of other users, who do not necessarily have to be direct friends of the owner of the photo. Additionally, users can be tagged in photos. These are all starting points for further crawling. The same applies for groups; A group gives access to the profiles of all group members, photos with users tagged, who are potentially not members of the group, and so forth. Consequently, a social snapshot of a single user does not only obtain the user's data and data of her friends, but its depth can reach a high value. Thus, the depth of the social snapshot is an essential configuration option which controls the social snapshot's extent. Figure 2 shows an example of a social snapshot with $depth = 2$. For a given user all of her friends are first fetched, followed by the friend's photos. The single path for photos of the friend's user illustrates the magnitude of available paths and thus data. Defining a specific social snapshot depth enables us to limit the amount of fetched data. The amount of data grows exponentially with social snapshot depth.

It is important to note that the relevance of data is not the same for different elements. For example, tagged users in a photo are most likely in a closer relationship to the owner of the photo than two users that joined the same group, just because of similar interests. Therefore, the social snapshot tool prioritizes element types that suggest higher data relevance and uses them as a starting point of each iteration. The prioritization is performed on the basis of predefined priority flags in the third-party application.

3.4 Modules

Our social snapshot application consists of a number of modules, which we describe in the following. The core modules are the automated web browser and our custom third-party application as outlined in Figure 1.

Social snapshot client. The social snapshot client module initializes the data gathering process with a given user's credentials or cookies. Once started, the client first authenticates itself against the target online social network. In the following, the client automatically adds our custom third-party application with the highest possible permissions to the target's account. Information that cannot be retrieved through our third-party application is crawled and parsed by the client. Once all information has been retrieved, the client removes the third-party application and logs out of the given social networking account. The interaction with the social network as well as web-crawling is performed by the Selenium framework [22], which we describe in the following. We implemented the social snapshot client in Java and the module offers a command line interface.

Automated web browser. The browser module is responsible for the basic interaction with the target online social network. We used the Selenium testing framework [22] to automate the Mozilla Firefox browser. Selenium comes with a command line server that receives Selenium commands. Therefore, we can use the framework to script the behavior of an average user using her Firefox web-browser to surf a social networking website. We had to overcome one initial obstacle though: cookie authentication with Selenium which was not supported out-of-the-box. We finally patched the original Java source code of the command line server to be able to correctly set HTTP cookies for the cookie authentication mode.

Third-party social snapshot application. Our OSN social snapshot application is a third-party application, which sole purpose consists of gathering all possible account data through the target OSN's API. The main design goal of our third-party OSN application is performance, thus multiple program threads are used to gather information as quickly as possible. The third-party application can be configured to prioritize specific account data and to download only a predefined set of account artifacts (social snapshot depth).

Hijack. The hijack module is a network sniffer module that collects valid OSN HTTP authentication cookies from sources such as LAN or WiFi connections. We built our hijack module on the basis of Mike Perry's modified libpkt library[23], which works out of the box with LAN, unencrypted WiFi, and WEP encrypted WiFi connections. The hijack module offers a command line interface and is implemented in Python.

Digital image forensics. The digital image forensics module matches image files gathered from online social networks with their original source. The goal is to find the pristine image of a compressed picture extracted through our social snapshot application. All images are initially clustered according to their color histograms, rescaled and compressed to the target picture size, and finally matched with pattern recognition techniques. As social networks typically remove meta (EXIF) information of uploaded images this module is helpful in finding the source of collected pictures from OSNs and thus restore information such as the original image creation time, camera model etc.

Analysis. The analysis module is a parser for the results gathered with the data collection modules of our application. It parses the crawled data as well as the information collected through the OSN's API. Furthermore, the analysis module fetches additional content such as photos that are openly available by knowing the URI from online social networks. Finally, it generates a report on the social snapshot data. The analysis module can be used to generate exact timelines of communication, metadata summaries, e.g. of pictures, a weighted graph from the network of friends, or their online communication.

4. RESULTS AND EVALUATION

In this section, we describe the evaluation of our social snapshot application. Our generic social snapshot approach is applicable to the majority of today's social networking services. The sole requirement for target social networks is the availability of a developer API or the adaption of our automated browser.

For a forensic tool there are some special requirements:

- Ability to *reproduce* results,
- Create a *complete* snapshot of the account.

To make digital evidence sufficiently reliable for court it is helpful if the process of gathering the evidence can be reproduced with identical results. In dynamic Web-based applications this is not possible because data is continuously added (eg. posts by friends) or removed (eg. friends-of-friends deciding to unshare data by modifying their privacy settings). It is, however, possible to have two or more independent investigators make snapshots at a similar time. While not all artifacts will be identical one can easily compare the sets of artifacts retrieved by our tool.

It is important that all artifacts used in the case are contained in both sets and that the sets do not contain too many unique artifacts because this would suggest that the snapshots are not reliable. Similar to information retrieval research we can thus adapt the metrics of precision and recall. n independent investigators gather each a set of artifacts A_i.

$\text{Precision}_j = \frac{|\bigcup_{i=0}^{n} A_i \cap A_j|}{|A_j|}$ $\text{Recall}_j = \frac{(|\bigcup_{i=0}^{n} A_i) \cap A_j|}{|\bigcup_{i=0}^{n} A_i|}$. Both can be combined to the F score.

$$F = 2 \cdot \frac{\frac{|\bigcup_{i=0}^{n} A_i \cap A_j|}{|A_j|} \cdot \frac{(|\bigcup_{i=0}^{n} A_i) \cap A_j|}{|\bigcup_{i=0}^{n} A_i|}}{\frac{|\bigcup_{i=0}^{n} A_i \cap A_j|}{|A_j|} + \frac{(|\bigcup_{i=0}^{n} A_i) \cap A_j|}{|\bigcup_{i=0}^{n} A_i|}} \tag{1}$$

4.1 Social Snapshots on Facebook

At the time of writing Facebook is the most popular online social network with a claimed user base of over 600 millions of users. Furthermore, Facebook supports third-party applications and user profiles contain a plethora of information. We thus decided to evaluate our social snapshot tool on Facebook. Third-party applications on Facebook have access to account data via the Graph API[10]. Almost the entire account data of Facebook users and their contacts are made available through their API. Facebook solely makes sensitive contact information such as phone numbers and e-mail addresses inaccessible to third-party applications. Hence our social snapshot client crawls the contact information of Facebook profiles, while all remaining social data is fetched through a custom third-party application. In October 2010, Facebook introduced a download option[12] that enables users to export their account data. Table 1 outlines the different profile content elements gathered through our social snapshot application as compared with Facebook's download functionality. As shown in Table 1, the download functionality only offers a very limited representation of a user's online activity. For example, for a given user's friends, only their ambiguous names are made available and no information on the activity of a given user's friends is included.

Element	**Download**	**social snapshot**
Contact details	–	✓Crawler
News feed	–	✓Graph API
Checkins	–	✓Graph API
Photo Tags	–	✓Graph API
Video Tags	–	✓Graph API
Friends	name only[a]	✓Graph API
Likes	name only[a]	✓Graph API
Movies	name only[a]	✓Graph API
Music	name only[a]	✓Graph API
Books	name only[a]	✓Graph API
Groups	name only[a]	✓Graph API
Profile feed (Wall)	limited[b]	✓Graph API
Photo Albums	limited[b]	✓Graph API
Video Uploads	limited[b]	✓Graph API
Messages	limited[b]	✓Graph API

[a] No additional information available.
[b] Missing meta-information such as UIDs.

Table 1: Account information available through social snapshots compared with Facebook's download functionality.

4.2 Hardware and Software Setup

To test the functionality of our social snapshot application, we developed a third-party application for Facebook based on their PHP Graph SDK. One of the main modifications we performed on their original library was the support for multi-threaded API requests. Our third-party social snapshot application for Facebook is thus able to handle a number of predefined API requests simultaneously. The single requests are hereby pushed on a request queue with a specific priority. Hence our third-party application can be configured to, for example, fetch private messages before user comments of a Facebook group. The extent/depth of social snapshots can be further configured as a parameter for our third-party application. We deployed it on a Linux server in our university network.

Our third-party application fetches Facebook elements of a given account and stores them as separate JSON files. The separate JSON files correspond to specific requests, whereas the files are named as follows. The first part of the JSON file name is the ID of an API object while the second part specifies the requested connection detail. For instance, "123456789~friends.request" contains all friends of the object with ID 123456789 formatted as a JSON object. In order to improve the performance of our application, we configured it not to download any videos or photos through the Graph API directly. As the third-party application collects direct links to photos, the digital image forensics module was configured to download photos during the analysis phase. Once the third-party application is finished fetching account data, it creates a tarball containing the social snapshot data.

The social snapshot client was adapted to fetch contact details of given user profiles and automatically add our third-party application to a target account. One particular challenge we had to overcome was to reliably obtain the list of friends of a given target account. Obstacles we had to cope with were the changing layout of the friend lists as well as Facebook only displaying a random subset of friends at a given time. We overcame the obstacles of creating the list of friends to be crawled, by fetching it through our third-party application and sending the profile links back to the client. Our client generates requests for every friend of the target user and sends them to the Selenium server that automates a Mozilla Firefox browser. The responses from the automated web browser module are parsed by the client and the contact information is extracted with a set of XPath queries. The client finally creates a CSV-file containing the contact information of all users. We deployed our client application in a virtual machine with a standard Ubuntu Desktop that runs our patched Selenium server. Our social snapshot analysis module implements both a parser for the fetched JSON Graph API requests as well as for fetched CSV contact details. The analysis module merges the results from the social snapshot client and the third-party application into a single database. We implemented the analysis module in Java.

We furthermore extended our digital image forensics module to automatically search a social snapshot for photo links, which it automatically downloads from the Facebook content distribution network. The hijack module did not require any Facebook specific modifications as it simply strips cookies of a given domain from a monitored network connection.

4.3 Test Subjects and Setting

We recruited human volunteers via e-mail, describing our experiment setting. The e-mail contained the experiment instructions and a briefing on how their personal information is going to be stored and analyzed. Furthermore, we briefed volunteers on the ethics of our experiment: no Facebook account data is modified, the social snapshots are stored in an encrypted filecontainer, no personal information is given to third-parties nor published. The invitation to support this first social snapshot evaluation was sent to researchers and students in computer science. Finally 25 people gave their consent to temporarily provide us access to their Facebook accounts. Volunteers temporarily reset their Facebook account credentials, which we used to create a social snapshot of their accounts. Once a social snapshot had been created, we informed our test group to reset their account password.

We configured our third-party social snapshot application for fetching an extensive account snapshot. We found that 350 simultaneous API requests lead to the best performance results in a series of indicative experiments we conducted beforehand. Our third-party application was configured to fetch the following elements recursively:

- Highest priority ($priority = 3$)
  inbox, outbox, friends, home, feed, photos, albums, statuses
- Medium priority ($priority = 2$)
  tagged, notes, posts, links, groups, videos, events
- Lowest priority ($priority = 1$)
  activities, interests, music, books, movies, television, likes

Our priority settings ensure that important information is fetched first. Account elements with highest and medium priority are fetched with $depth = 2$ while elements with the lowest priority are gathered with $depth = 1$. Thus a social snapshot of a given user includes for example, her friend's groups, tagged pictures, links etc. but no pictures, comments, etc. are downloaded from her favorite television series. These social snapshot settings imply that not only the target's account is completely fetched but also social data on the targets' friends is collected.

4.4 Results on Social Snapshot Performance

Figure 3 illustrates the time required by our third-party social snapshot application to snapshot the test accounts through the Graph API. Our third-party application required on average 12.79 minutes. Account elements of our test accounts were on average fetched with $93.1kB$ per second.

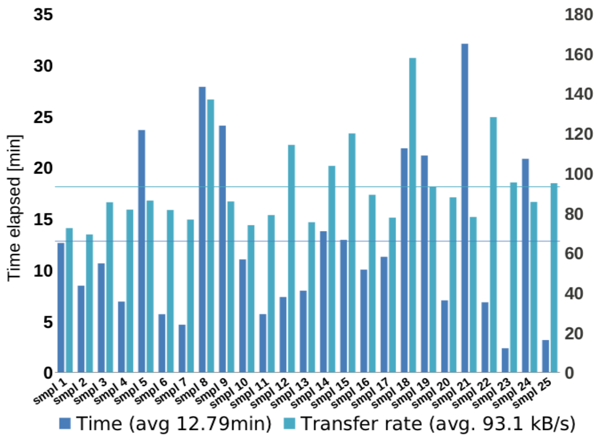


Figure 3: Required time and transfer rate of our social snapshot third-party application.

The time required for crawling contact details with our automated web browser is outlined in Figure 4. Test accounts have been crawled within 14 minutes on average. The average elapsed time per account corresponds to 3.4 seconds per user profile page.

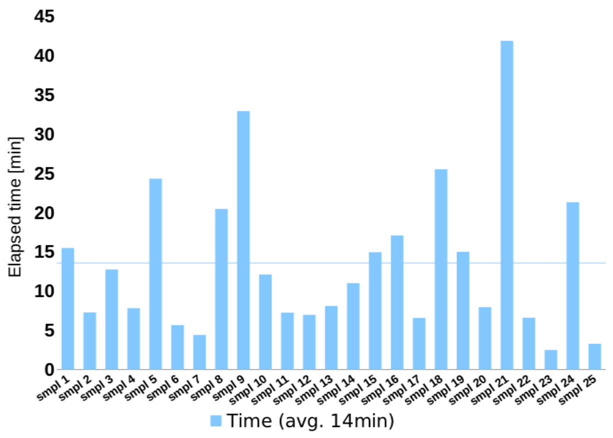


Figure 4: Time required for crawling contact details with social snapshot client and automated web browser.

4.5 Results on Social Snapshot Completeness

As illustrated in Figure 5, our third-party application found and fetched on average 9,802 Facebook account elements per test subject. The storage size of the fetched JSON files accounted to $72.29MB$ on average. Listing 1 shows an anonymized example from the fetched Facebook account elements. The example represents the basic information fetched of the user "John Doe" formatted as a JSON object. This example request also highlights that account data fetched through the Graph API provides a richer information set for further investigations. The standard web interface does not provide information if a user's account is verified nor an update time that is accurate to the nearest second with information on the used time zone.

Listing 1: Example of collected JSON element

```
{"id":"12345678","name":"John Doe",
"first_name":"John","last_name":"Doe",
"link":"http:\/\/www.facebook.com\/johndoe",
"username":"johndoe","birthday":"04\/01\/1975",
"hometown":{"id":"","name":null},
"quotes":"social snapshot your account!.\n",
"gender":"male","email":"johndoe@example.com",
"timezone":2,"locale":"en_US","verified":true,
"updated_time":"2011-05-15T13:05:19+0000"}
```

Compared to data collected via the standard web interface, our social snapshot contains a number of additional information tokens. Most notably for forensic investigation is the availability of exact creation timestamps through the Graph API. We used our image forensic module to download all unique photos in the highest available resolution from the gathered social snapshots. The downloaded photos corresponded to 3,250 files or $225.28MB$ on average per test account.

Figure 6 shows the additional contact details crawled with our social snapshot client. On average, our social snapshot client had to crawl 238 profile sites per test account. For all crawled profile pages our crawler found 22 phone numbers, 65 instant messaging accounts, as well as 162 e-mail addresses on average. We noticed that after a number of subsequent requests to user profiles of a given account, Facebook

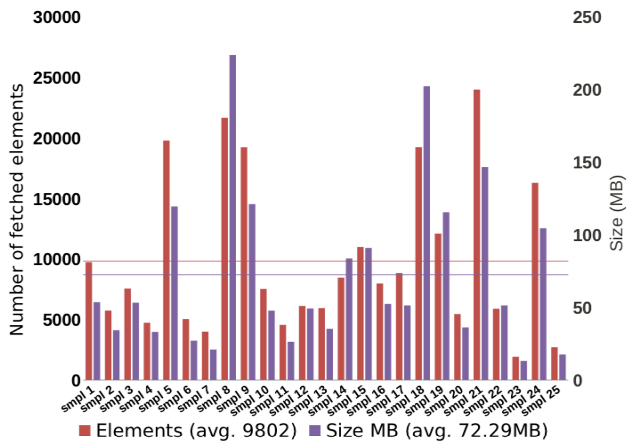


Figure 5: Account elements fetched through social snapshot third-party application.

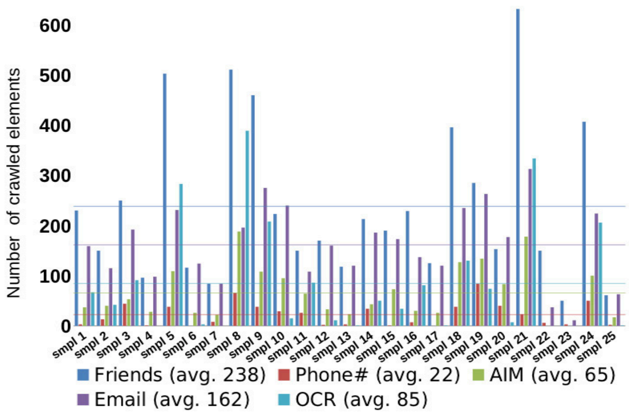


Figure 6: Contact details crawled with social snapshot client and automated web browser.

replaces textual e-mail addresses with images. This behavior was noticeable with our social snapshot client, whereas on average we fetched 85 e-mail addresses in image form (OCR in Figure 6). Due to the fact that Facebook uses e-mail addresses in image form as a web crawler protection method, we could not directly parse the fetched images.

Finally we used our analysis module to verify the integrity of the collected snapshots. We successfully verified that every entry in our fetched contact details CSV files had correspondent entries within the retrieve JSON files, as well as that no invalid responses where received through the Graph API. We furthermore implemented a mechanism for the analysis module to overcome the obstacle of parsing image e-mail addresses. By providing Facebook's e-mail image creation script the maximum possible font size of 35 instead the default of 8.7, we fetched higher resolution versions of the e-mail address pictures. We could thus rely on GNU Ocrad[14] to resolve these high resolution images into their textual representation. The idea of replacing the default font size with a larger one was first described in [25] and we could successfully verify that the described method still applies.

4.6 Indicative Cookie Authentication Experiments

We performed a number of indicative experiments to verify our cookie authentication method on Facebook. Both non-persistent as well as persistent cookie authentication is available. Persistent cookies are valid for 30 days in the case of Facebook. We successfully tested our social snapshot tool with the hijack module on a number of non-persistent users over an unencrypted test WiFi network. Furthermore, we successfully validated our social snapshot application with persistent cookies extracted from web browser profile files. In the case of one particular test setting, namely our university campus WiFi, we could observe as many as 50 valid social networking sessions within one hour.

4.7 Forensic Analysis of Social Snapshots

Collected social snapshots enable the forensic analysis of social network activity of specific users and their online peers. Since the entire content of a users' social networking account with exact timestamps is collected, timelines can be easily generated. Moreover, social snapshots offer a valuable source for further investigations. The collected e-mail addresses could for example be used to identify users on other online platforms such as photo and file storage services, while collected media data could be matched with evidence collected through traditional forensic images. Figure 7 shows an example of a generated timeline for a fictitious forensic investigation on the "Dalton Gang". The Dalton gang is suspected of having committed an aggravated bank robbery between 8:00am and 8:30am on the 13th of January 2011. All four gang members have an alibi for the specific time and said they were all on a joint getaway together with their families. Bob Dalton, the head of the gang, presents a group photo he posted on Facebook that very day. In order to validate the posting, five close friends of Bob give their consent to *social snapshot* their social networking accounts. While the posted group photo correctly shows up with the specified date in all five social snapshots, an interesting posting from Bob Dalton's wife is collected in two of the social snapshots. The posting dated one week before the robbery, timestamped with 01/06/2011 07:32:12 AM reads "Off to the beach, for our family group picture. Hehe". The investigators at this point start to suspect that the alibi picture had been taken a week beforehand to fabricate an alibi. Unaware to Bob's brother Grat Dalton, investigators social snapshot his account using the *hijack module* during his daily Internet browsing, exploiting a coffeeshop's insecure WiFi connection. Analyzing Grat's social snapshot the investigator noticed that Grat exchanged private messages with his brother Bob on the day of the robbery. The first messages with ID 00000000 sent at 3:20:32 PM reads "Grat, That was almost too easy today ... we should start thinking on how to spend all the Benjamins:-). greetings Bob". In the second message Grat replied to Bob at 6:27:12 PM: "Yeah almost too easy:-) Great idea with the group picture at the beach btw, that will cause them some serious teeth gnashing." With this further evidence on a possible false alibi, the investigators perform a house search on Bob Dalton's home. While the search does not reveal any of the stolen money,

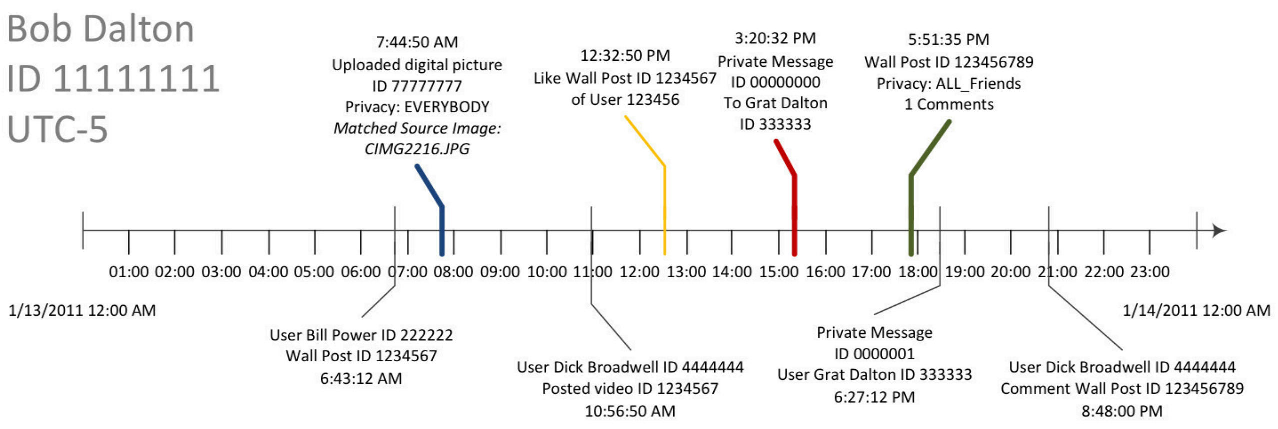


Figure 7: Example timeline created from collected social snapshot.

the personal computer of Bob Dalton is seized during the house search. Amongst digital documents and images the investigators find a valid Facebook *authentication cookie* on Bob's forensic image. The investigator creates a social snapshot of Bob's social networking account using the extracted authentication cookie. Comparing Bob's and Grat's social networking activity on the day of the robbery they find that the social snapshots accurately correlate with a F1-score of 0.84, and both accounts hold the treasonous private messages. The timeline generated from the social snapshot and outlined in Figure 7 shows Bob's online activity on the day of the bank robbery. Curious as to whether the pristine digital image of Bob's posting can be recovered the investigator runs the *digital image forensic module* to match digital images from the forensic image with the image collected through the seven independent social snapshots. The digital image forensic module reports a positive match on a digital image named "CIMG2216.JPG". The original EXIF information of image "CIMG2216.JPG" reveals that their alibi group picture had indeed been taken a week before the robbery.

4.8 Social Snapshot Open-Source Release

We release the social snapshot core framework for Facebook under a GPL v3 open source license[1]. The source code contains the social snapshot client, our third-party application, as well as the patched Selenium server. Not included in the open source release are the analysis and photo forensics modules. We furthermore decided not to release the hijack module, which could be potentially misused for malicious attacks.

5. DISCUSSION

Our evaluation required on average 9,802 API and 238 HTTP requests to successfully snapshot an entire social networking account in less than 15 minutes. In order to collect forensic evidence with traditional web-crawling more than 10,000 HTTP requests are necessary to snapshot a single test account. The generated network traffic of traditional web-crawling would have been likely detected and blocked by social networking providers. Moreover, our evaluated approach retrieved the great majority of social networking account data without the requirement of additional parsing and with exact timestamps. During the implementation of our social snapshot techniques, Facebook's web-site layout changed a number of times. Since only contact details were crawled, we could promptly adapt the parser of our client, while our third-party application did not require any changes at all. As Facebook has no review process for third-party applications we could also make our third-party application available straightforward. Third-party applications on Facebook do not even have to appear in their application directory in order to be usable.

Apart from digital forensics, social snapshots could also be used to raise user awareness. Users would run our social snapshot tool and get a report on their account data. Thus, social networking users could sight the magnitude of information that is stored with their social networking providers. We hope that this would help the average social networking user to make better informed decisions on which information they post.

Unencrypted social networking sessions enable the gathering of social snapshots for digital forensics but also pose a serious security threat. Since HTTPS is not enabled by default on today's social networking services, user sessions can easily be hijacked. Two proof-of-concept tools have been released that make session hijacking of social networking sessions available to the average user. *Firesheep* [3] has been released in October 2010 as a browser extension and at the time of writing is not functioning anymore. *Faceniff* [24] offers a point-to-click interface and supports a number of wireless network protocols. It is an Android application for hijacking social networking sessions released in June 2011. Both hijacking applications were released in order to create awareness for the problem of insecure social networking sessions. It is trivial however to couple such simple hijacking applications with our social snapshot tool. Thus, attackers could harvest complete account snapshots in an automated fashion. It has been shown [17] that the large amount of sensitive data stored in social networks could be used for large-scale spam attacks via session hijacking.

6. RELATED WORK

Numerous forensic frameworks have been proposed in re-

[1]`https://github.com/mleithner/SocialSnapshot`

cent years. However, none of them were designed specifically to extract information from social networks. To the best of our knowledge, no other publication has examined the impact of a hybrid API and crawler based approach to digital forensics in social networks.

Even though social networks are not per-se part of the cloud computing paradigm, the area of cloud forensics poses some related challenges as these service operators rely on private clouds for their infrastructure. Specifically the unknown location of data centers [26] and the difficulty to obtain access to forensic data sources without trusting a third party [2] as well as data provenance [20]. Pyflag [8], on the other hand, is a modular network forensic framework built to analyze network dumps. Among other features it is able to rebuild HTML pages from packets, allowing the examiner to view the webpages the suspect has seen even if it used AJAX or other dynamic techniques for representation. Xplico [30] is an Internet traffic decoder which can retrieve Facebook chat conversations from network dumps.

In relation to our digital image forensics module a recent approach is PhotoDNA [21], which is a program to detect known and explicitly illegal pictures based on calculated signatures. It is only available to law enforcement agencies. Similar to signature-based antivirus software, a trusted party calculates the signatures for illicit pictures such as child pornography which in turn is then compared with the signatures of pictures in webpages, data archives or pictures from forensic hard drive examinations. In [19] characteristics of embedded thumbnails are used to authenticate the source of a picture. While both approaches work similar to our module, they have not been designed or employed to compare digital images from social networks with pictures from a suspect's hard drive.

7. CONCLUDING REMARKS

Social snapshots explore novel techniques for automated collection of digital evidence from social networking services. Compared with state-of-the-art web crawling techniques our approach significantly reduces network traffic, is easier to maintain, and has access to additional and hidden information. Extensive evaluation of our techniques have shown that they are practical and effective to collect the complete information of a given social networking account reasonably fast and without detection from social networking providers. We believe that our techniques can be used in cases where no legal cooperation with social networking providers exists. In order to provide a digital evidence collection tool for modern forensic investigations of social networking activities, we release our core social snapshot framework as open source software. We will continue to extend the analysis capabilities of our forensic software and cooperate with partners on the evaluation of real-world cases.

7.1 Acknowledgments

The research was funded by COMET K1, FFG - Austrian Research Promotion Agency, by the Austrian Research Promotion Agency under grants: 820854, 824709, 825747, and by the European Union Seventh Framework Programme (FP7/2007-2013) under grant agreement n° 257007. Recruiting test subjects in a security lab and computer science environment was especially challenging as most people misconceived our temporary Facebook access request for a clumsy social engineering attack. Therefore, we would like to especially thank the human volunteers who supported our experiments by providing their Facebook data. The authors would also like to thank Barbara Weber and Robert Sablatnig for their advice and feedback in the creation of this paper.

8. REFERENCES

[1] L. Bilge, T. Strufe, D. Balzarotti, and E. Kirda. All your contacts are belong to us: automated identity theft attacks on social networks. In *Proceedings of the 18th international conference on World wide web*, pages 551–560. ACM, 2009.

[2] D. Birk and C. Wegener. Technical issues of forensic investigatinos in cloud computing environments. In *Systematic Approaches to Digital Forensic Engineering, 2011. SADFE 2011. Sixth International Workshop on*. IEEE.

[3] E. Butler. Firesheep. Online at `http://codebutler.com/firesheep`, oct 2010.

[4] M. Caloyannides, N. Memon, and W. Venema. Digital forensics. *Security & Privacy, IEEE*, 7(2):16–17, 2009.

[5] B. Carrier. *File system forensic analysis*. Addison-Wesley Professional, 2005.

[6] E. Chan, S. Venkataraman, F. David, A. Chaugule, and R. Campbell. Forenscope: A framework for live forensics. In *Proceedings of the 26th Annual Computer Security Applications Conference*, pages 307–316. ACM, 2010.

[7] CNN. Facebook status update provides alibi. Online at `http://cnn.com/2009/CRIME/11/12/facebook.alibi/index.html`, nov 2009.

[8] M. Cohen. PyFlag-An advanced network forensic framework. *digital investigation*, 5:S112–S120, 2008.

[9] EFF. Social Media and Law Enforcement: Who Gets What Data and When? Online at https://www.eff.org/deeplinks/2011/01/social-media-and-law-enforcement-who-gets-what.

[10] Facebook. Graph API. Online at `https://developers.facebook.com/docs/reference/api/`.

[11] Facebook. Statistics of Facebook. Online at `http://www.facebook.com/press/info.php?statistics`. Accessed April 20th, 2011.

[12] Facebook. The Facebook Blog: Giving You More Control. Online at `https://blog.facebook.com/blog.php?post=434691727130`, oct 2010.

[13] K. Fowler. *SQL Server forensic analysis*. Addison-Wesley Professional, 2008.

[14] FSF. Ocrad - The GNU OCR. Online at `http://www.gnu.org/software/ocrad/`.

[15] H. Gao, J. Hu, C. Wilson, Z. Li, Y. Chen, and B. Zhao. Detecting and characterizing social spam campaigns. In *Proceedings of the 10th annual conference on Internet measurement*, pages 35–47. ACM, 2010.

[16] B. Hay, K. Nance, and M. Bishop. Live analysis: Progress and challenges. *Security & Privacy, IEEE*, 7(2):30–37, 2009.

[17] M. Huber, M. Mulazzani, E. Weippl, G. Kitzler, and S. Goluch. Friend-in-the-middle attacks: Exploiting social networking sites for spam. *Internet Computing*, 2011.

[18] T. Jagatic, N. Johnson, M. Jakobsson, and F. Menczer. Social phishing. *Communications of the ACM*, 50(10):94–100, 2007.

[19] E. Kee and H. Farid. Digital image authentication from thumbnails. *Proceedings of the SPIE, Electronic Imaging, Media Forensics and Security XII*, 2010.

[20] R. Lu, X. Lin, X. Liang, and X. Shen. Secure provenance: the essential of bread and butter of data forensics in cloud computing. In *Proceedings of the 5th ACM Symposium on Information, Computer and Communications Security*, pages 282–292. ACM, 2010.

[21] Microsoft. PhotoDNA. Online at `http://www.microsoftphotodna.com/`.

[22] OpenQA. Selenium wep application testing system. Online at `http://seleniumhq.org/`.

[23] M. Perry. CookieMonster: Cookie Hijacking. Online at `http://fscked.org/projects/cookiemonster`, aug 2008.

[24] B. Ponurkiewicz. Faceniff. Online at `http://faceniff.ponury.net/`, jun 2011.

[25] N. A. Rahman. Scraping facebook email addresses. Online at `http://www.kudanai.com/2008/10/scraping-facebook-email-addresses.html`, aug 2008.

[26] M. Taylor, J. Haggerty, D. Gresty, and D. Lamb. Forensic investigation of cloud computing systems. *Network Security*, 2011(3):4–10, 2011.

[27] The New York Criminal Law Blog. Criminal found via Facebook. Online at `http://newyorkcriminallawyersblog.com/2010/03/assault-criminal-who-was-found-via-facebook-is-back-in-ny.html`, mar 2009.

[28] The Washington Post. Facebook: a place to meet, gossip, share photos of stolen goods. Online at `http://www.washingtonpost.com/wp-dyn/content/article/2010/12/14/AR2010121407423_pf.html`, dec 2010.

[29] G. Wondracek, T. Holz, E. Kirda, and C. Kruegel. A Practical Attack to De-Anonymize Social Network Users. In *Proceedings of the IEEE Symposium on Security and Privacy*, 2010.

[30] Xplico. Xplico - Network Forensic Analysis Tool. Online at `http://www.xplico.org/`.

A Peel of Onion

Paul Syverson
Center for High Assurance Computer Systems
U.S. Naval Research Laboratory
Washington DC, USA
syverson@itd.nrl.navy.mil

ABSTRACT

Onion routing was invented more than fifteen years ago to separate identification from routing in network communication. Since that time there has been much design, analysis, and deployment of onion routing systems. This has been accompanied by much confusion about what these systems do, what security they provide, how they work, who built them, and even what they are called. Here I give an overview of onion routing from its earliest conception to some of the latest research, including the design and use of Tor, a global onion routing network with about a half million users on any given day.

1. WHY ONION ROUTING

We (David Goldschlag, Michael Reed, and I) began work on onion routing in late 1995 with the goal to separate identification from routing. Onion routing has often been said (by ourselves and others) to provide anonymous communication, but 'anonymity' can be taken in many ways, and misunderstanding about that has led to significant missteps in system design and analysis. To quote from our paper at ACSAC '96, "Our motivation here is not to provide anonymous communication, but to separate identification from routing. Authenticating information must be carried in the data stream. Applications can (and usually should) identify themselves to each other. But, the use of a public network should not automatically reveal the identities of communicating parties. The goal here is anonymous routing, not anonymity." [43].

If I need to log into the workstation in my office at the Naval Research Laboratory from a remote location, I want to be sure that I am connecting to the right system before I start entering my username and password. Similarly, I want the system to be sure it's me before granting access. Nonetheless, I may not want, e.g., the wireless access point at my remote location or anyone in range of it to know that someone is connecting to NRL from there. Nor might I want this to be known by all the other elements on the communication path from me to my workstation.

In other cases, however, I may want to hide my identity from the system at the far end. For example, if I am reviewing a submission for publication or a proposal from someone, where review is to be anonymous and I need to check something at the submitter's web site, I would like to do so without revealing my identity to the web server. What I mean by 'identity' we will revisit more thoroughly below. But if I am originating the connection at my office workstation, it might include my IP address as well as anything else that could indicate who the reviewer is or possibly broader aspects, such as that the reviewer is from the Navy. This could be indicated by either things I type or things that applications send through the connection, for example, cookies. But again, the goal is to separate the choice to identify or not from the routing necessary for communication to take place.

ACSAC '11, December 5–9, 2011, Orlando, Florida USA
2011 ACM 978-1-4503-0672-0/11/12.

2. WHO ONION ROUTING

Onion routing systems have a wide variety of users with a variety of purposes. Tor is the most well known and widely used onion routing system. As of the time of writing it serves over half a million concurrent users over a network infrastructure comprised of about three thousand onion routers of one type or another [53].

I have already described two ways users might want to protect themselves using onion routing. There are many types of users and many uses for onion routing. Law enforcement and intelligence agencies need to operate on the Internet without revealing their activities or intentions to those they are investigating or anyone else observing. And the "road warrior" mentioned above is protected both from profiling of his online activities and affiliation, and from potential physical harm by an adversary who can easily and discretely associate these with the hotel he is staying in [19]. Road warriors typically use Virtual Private Networks to help protect their communications. But VPNs provide limited protection against this threat if it is easy to associate users or affiliations with a VPN, and the adversary can observe connections to the VPN. Also, while onion routing distributes trust as we shall see, externally provided VPNs typically constitute a single point of trust and thus a single point of failure for protecting routing information.

Penetration testing engineers have found that if they have a limited IP space from which to launch attacks, defenders have been able to use this to spot attacks. Onion routing can help in this regard, although without additional safeguards this would make onion routing networks appear to be a vector for abuse [18].

Victims of various types of diseases, as well as of crimes and abuse have used onion routing to do research and to chat with fellow victims without risk of exposure [15]. Besides the more obvious information security protections these afford, the ability to speak freely and anonymously has long been recognized as an important therapeutic benefit.

Many ordinary citizens simply want to limit the amount of information about themselves that is being gathered every time they go online. How this information will be exploited by businesses or criminals is often not yet understood at the time it is gathered, so prudence counsels prevention. Or they may want to protect their children from being geolocated by those observing or engaging the children in online activity—geolocation that can take place simply from the act of connecting even before the child has entered anything into the application.

This is just a small sampling of the types of onion routing users. Central to the protection that onion routing provides is that all of these different types are sharing the network. Otherwise the problem noted above for VPNs would apply to onion routing as well. Anything coming out of, for example, a Navy-only onion routing network would be known to be coming from the Navy and anything entering it would be known to be headed to the Navy: this would not adequately separate identification from routing. But the diverse users needed to provide this protection also have diverse trust values. Thus the entire network infrastructure cannot be provided by or under the control of a single entity. And since those running the network will similarly have diverse trust, they must be able to examine for themselves the code that they run, or at least be sure that independents whom they trust can do so. These points were part of our vision for onion routing from the very beginning, and we obtained our first publication release for onion routing code in 1996, before 'open source' was a generally adopted concept.

Besides needing to adequately trust the system, both users and providers must want to use and/or provide the system. This was addressed in the papers cited above. But even before then, it was the central topic of "On the economics of anonymity" [1]. From the perspective of users, a basic dilemma arises from the difficulty of trying to be anonymous by yourself. Unlike other security properties, those who need strong protection will not be early adopters because the system will then be known to be protecting the relatively small set of most sensitive users. So merely connecting or sending to it will be revealing of something they will probably want to hide. Similarly any destination of traffic from the system is revealed as something of interest to one of those most sensitive users. Contrastingly, the less sensitive user will have inadequate incentive to pay for the system. This means that a certain degree of free riding must be built into the system. On the one hand, this means that less sensitive users will be getting protection virtually for free. On the other hand, by their use of the system they effectively provide better protection for the more sensitive users. So it is not free riding from the security perspective.

But even just looking at user cost, it is not really entirely for free. The security onion routing provides comes with some inevitable usability and performance overhead. We will get more into how it works below, but for now it is enough to note this and to note its impact on security. Back et al. have stated that "In anonymity systems usability, efficiency, reliability and cost become *security* objectives because they affect the size of user base which in turn affects the degree of anonymity it is possible to achieve." [4]. In fact perceived usability by others is also a factor because it affects expectations about how likely others are to use the system and provide us with cover. In a sense, a system that is not as secure by some analytic criterion may be more secure in fact if opinion drives more users to that system [15]. We will revisit such seemingly paradoxical issues below when we discuss security definitions and metrics.

3. HOW ONION ROUTING

Before getting into defining 'security', we should spend a little time defining 'onion routing' and describing how it works because there has been a surprising degree of confusion about what onion routing is and is not.

Symmetric-key cryptography is generally much cheaper computationally than public-key cryptography is. But symmetric-key cryptography only works if both communicating parties share a key. An advantage of public keys is that Alice can send an encrypted message to Bob without having to share a key with him first. A primary application of public-key cryptography is thus in protocols to establish a shared (symmetric) key for Alice and Bob to use in a communication session. If a large amount of data is sent in the session encrypted with this shared key, the computational overhead is much less than if public keys were used throughout the session. There are many other benefits to establishing a shared session key, but we focus on this feature for now.

Onion routing is in one sense just a generalization of this basic idea. It can use the more computationally expensive public-key cryptography to lay a cryptographic circuit of shared symmetric keys along an unpredictable route. The circuit can be bidirectional, and each session key is shared between the initiating client and one onion router in the path. Data that is sent by the client is wrapped in layers, except that the layers are created using the shared symmetric keys rather than public keys. Using the symmetric key it learned when the circuit was laid, each onion router removes a layer of encryption as the data passes. In this way the data emerges at the end of the circuit as plaintext (from the perspective of the onion routers. It could be ciphertext if that is what is being sent over the circuit by the application.) Data passing back from the responder, for example from a web server, has a layer of symmetric crypto added to it as it passes through each onion router node in the path. It thus arrives at the circuit initiator as a layered data structure. The circuit initiator set up the circuit and so is able to remove the layers using the symmetric keys for that circuit.

Onion routing was invented to facilitate anonymous low-latency bidirectional communication, such as occurs in web browsing, remote login, chat, and other interactive applications. By only using public-key cryptography to establish session keys it allows for throughput and latency that would not be feasible if public-key operations were needed for each message (or packet) passing through the system. By following a multihop free-route path selection through a network of independently managed onion routers, it makes it hard for an adversary to observe traffic entering and leaving the system.

3.1 An onion by any other name would smell as sweet

In onion routing, application data is passed inside a layered data structure as just described. In the first two generations of onion-routing design [27, 44, 50] that emerged from NRL, the circuit was also laid by means of a layered data structure, with two primary differences. (1) Each layer also provided the material to generate the symmetric keys used for passing the data back and forth. (2) The final layer contained neither a destination address nor meaningful content to be transmitted. Thus, abstractly the onion for a three-hop route through R_1, R_2, R_3 looked like this:

$$\mathrm{E}(\mathrm{PK}_{R_1}, [K_1, R_2, \mathrm{E}(\mathrm{PK}_{R_2}, [K_2, R_3, \mathrm{E}(\mathrm{PK}_{R_3}, [K_3, \mathrm{Pad}]\ )]\ )]\ )$$

where 'PK_{R_i}' is a public key for R_i, and 'K_i' is session-key material to be shared between the route originator and R_i. We are ignoring some details here to focus on these two features of the structure. In particular, note that what is encrypted for the last layer is just the symmetric-key material to be used for passing data back and forth and then just empty padding. Thus, there is no message at the innermost core of the onion: the onion is the layers themselves, much like the root vegetable from which it gets its name. Indeed, this structure is the basis for the choice of the name "onion routing": the network routes *onions*—not just layered data structures, but data structures that are comprised of layers and nothing else. Using "onion" to describe a layered data structure is not unique to onion routing or to structures specifically comprised of nothing but layers. Independently and contemporaneously with the first descriptions of onion routing, "onion" was used by Gülcü and Tsudik to describe a structure somewhat like the ones above for a mix-based[1] remailer network [28], and there may be even earlier uses. We are not averse ourselves to referring to the packets being sent over the cryptographic circuit as "data onions".

Furthermore, though the circuit-building onions gave onion routing its name, they are not what makes it onion routing. The essential feature is that public keys are used to lay a cryptographic circuit of symmetric keys, which is then used to pass data. Other onion routing designs that we will discuss later do this by effectively only sending one onion layer at a time through the partial circuit already created.

'Onion routing' has also at least occasionally been used in published research to refer to networks that do not lay a circuit at all. Though we will countenance somewhat varied use of 'onion' to cover "data onions" or "message onions", we will stick to using 'onion routing' for designs that lay cryptographic circuits—which was what those of us who coined the phrase had in mind.

It is instructive to explore the origins of this mistake in usage, but it requires that we examine the differences between mixes and onion routers. Mixes were invented by David Chaum in 1981 [9]. Briefly, a mix takes in a batch of packets or messages, reorders them, cryptographically changes their appearance, possibly adds new messages or previously delayed messages from an earlier batch, and possibly removes some received messages from the batch. It then forwards the resulting batch to other mixes or to ultimate destinations. The mix's operations make it difficult for anyone observing the mix to correlate observed input messages with observed output messages. Other than for purposes of contrast, we will not discuss mixes, which have many variations and an extensive literature in their own right [11, 21].

[1]We will say more about mixes presently.

Mix networks get their security from the mixing done by their component mixes, and may or may not use route unpredictability to enhance security. Onion routing networks primarily get their security from choosing unpredictable routes through a network, and onion routers typically employ no mixing at all. This gets at the essence of the two even if it is a bit too quick on both sides. Other typical and highly salient distinctions include that all existing onion routing network designs are for carrying bidirectional low-latency traffic over cryptographic circuits while public mixnets are designed for carrying unidirectional high-latency traffic in connectionless messages[2]. Mixes are also usually intended to resist an adversary that can observe all traffic everywhere and, in some threat models, to actively change traffic. Onion routing networks are generally completely broken against an adversary who observes both ends of a communication path. Thus, onion routing networks are designed to resist a local adversary, one that can only see a subset of the network and the traffic on it.

Given the fundamental differences in the mechanisms they employ, the adversaries they are intended to resist, and their basic designs (not to mention typical applications), how could anyone who works in this area ever confuse the two? All deployed onion routing networks do use some form of layered encryption on traffic they carry, encryption that is gradually removed as it passes through the network. And this is also true of decryption mixnets (re-encryption mixes work differently). Thus there is a clear similarity between the two in at least this respect. Still, given the differences, how could this be enough to confuse an expert? The standard conception of security found in the literature—which in turn motivates system designs—assumes the goal of all systems that hide who is talking to whom is to make a given set of users (or user communications) less distinguishable. If one motivates design by starting with such a set and seeing how well the system obscures identification of its elements, the security contributions of an onion routing approach are harder to see. Any distinction between onion routing networks and mixnets, if recognized at all, is then likely to be couched only in terms of differences in intended application or engineering tradeoffs of security versus performance. Even we designers of onion routing systems have been occasionally guilty of falling into this idiom. We discuss adversaries and security definitions further in [48] and below in section 5. Our goal here was simply to make clear the difference between mix networks and onion routing networks.

Another confusion is that we have historically used 'onion routing' both to refer to the general approach described above and to refer to specific system designs that emerged from projects at NRL. To distinguish between these below, we will use 'onion routing' to refer to all onion routing systems and will use 'NRL onion routing' to refer to the specific designed and deployed versions of onion routing that have come out of the NRL onion routing projects, unless this is clear from context. References to one generation or another of onion routing will also be an indication that it is specifically one of the NRL-originated designs that is being discussed.

[2]An exception is Web MIXes [6], which creates bidirectional circuits through mix cascades to carry public web traffic.

Though onion routers are not mixes, it is possible to have a combination, producing a subclass of onion routers (and of mixes). Specifically, in the second-generation design from NRL, some low-latency timed mixing was added [44]. Prior anonymizing networks had been based on mixing, and realtime mixing per se did not add much overhead. It was hoped that by introducing realtime mixing, it would be possible to explore its possible benefits and simplify security analysis at least with respect to an external passive attacker. It was also hoped that this would make onion routing more palatable to the community by making it more familiar. If mixing was to be anywhere near realtime, however, then it would be necessary to do some padding and/or rate limiting to avoid trivial timing correlations. Though a full-length padding scheme was immediately dismissed as too costly, padding and bandwidth limiting based on a sliding-window weighted average of prior traffic was considered potentially viable [51]. Just like the first and the third, this second-generation design of onion routing made use of TCP and stream ciphers on the links between onion routers as well as end to end. The mixing onion routers had to be low-latency, mixing only packets that had arrived during a very short prior interval. But, the use of TCP and stream ciphers also meant that mixing could only be of packets on different circuits going through an onion router: traffic on a given circuit was not mixed with itself so as to preserve order. (TCP (Transmission Control Protocol) is one of the two main ways that traffic is passed over the Internet. It provides guaranteed and in-order packet delivery at the expense of some timeliness. The other main protocol is UDP (User Datagram Protocol), which provides no such guarantee and is better for applications with tighter realtime constraints that can tolerate some lossiness, for example VOIP.) Though padding or other approaches might prevent most short-term passive eavesdropping, nothing practical that has been proposed to date is expected to be effective against an active attacker. Various schemes have been proposed to cope with active timing correlation attacks. The earliest was PipeNet, which was originally proposed in a cypherpunks mailing list post in 1995 [10], although it was never formally published. PipeNet essentially required that all sending be at a persistent constant rate. To fully avoid any correlation no sender or recipient can join or leave the network once it is formed, and the entire network must be shut down as a single originator of communication stops sending. Though not all subsequent proposals are as draconian, and even ignoring the padding overhead, those that aren't are generally ineffective: they impose too much delay, require that the responder be an active participant in the scheme, or some combination of these.

By the time the third-generation onion routing design was begun, nobody had proposed a practical traffic-shaping scheme that wasn't fundamentally weak in some way. Indeed that remains true to this day. An unknown research breakthrough remains possible; however, all thought of making onion routers function as mixes was abandoned when the third-generation design was begun until such time as this could be shown to have any value against realistic adversaries [17].

4. WHICH ONION ROUTING

This paper was invited to accompany an ACSAC classic paper talk. But, I must confess that I am not sure which classic onion routing paper prompted the invitation. Within a half year of publishing the first onion routing design at the Information Hiding Workshop in May of 1996 [27], we published another paper at ACSAC that first described separating a client proxy that constructs the circuit from the onion routers that can carry circuits from many clients [43], and the following May we presented another paper that introduced many of the second-generation features most closely associated with onion routing [52].

I have alluded above to three generations of onion routing. I mean by this the three generations of onion routing design done by U.S. Naval Research Laboratory (NRL) researchers and various people contracted to work with them—most notably for the third-generation (Tor), Roger Dingledine and Nick Mathewson of what is now the Tor Project. There are many differences between the generations and difference from other flavors of onion routing besides those that came from NRL. Most of these we will ignore for now, but some of the primary distinctions between the three generations I mentioned are the following.

First-generation onion routing:

- **fixed-length, five-node circuits:** Onion routing was intended to allow anonymous connections to Internet servers, such as web servers, that were not part of the onion routing network. But, a configuration that was also intended for protection was communication between two enclave firewalls, on both of which were onion routers. If routes were three hops, a hostile middle node would immediately learn both source and destination enclaves. If routes were four hops, someone compromising the node adjacent to the source or destination enclave would immediately know which other node to attack to learn an entire circuit and thus who was talking to whom. By having five hops in a circuit this adversary would not have this information unless it attacked another node first.

- **integrated configuration:** A client, as described above, and an onion router are fully combined. Client applications could proxy through onion routing without participating in the network or running any software. But, all elements of onion routing communication are essentially between peers, and building of routes is entirely determined by and under the control of the first onion router in the circuit.

- **static topology:** There were no provisions for topology change or network discovery in the first-generation design. It was assumed that there would be a modest size network of nodes (perhaps twenty to one hundred) run by stable organizations that were not all mutually trusting of each other and that information about network configuration, signing keys, etc. would be handled offline. This was thus seen as at most a detail to be addressed later.

- **loose-source routing:** In case a node was unable to connect to the subsequent node in a circuit, e.g., because of underlying network problems, it could build its own onion through which it would pass the onion it was given. Thus, it would attempt to complete the circuit by adding more indirection to the route.

- **numerous application-specific proxies:** In the mid nineteen nineties applications were increasingly written to be proxy-aware as firewalls became more common; however, most applications did not yet use the SOCKS protocol that standardizes such interaction. For this reason, the first-generation of onion routing contained specific proxies for web traffic, for remote login, for email, and for other applications.

- **rendezvous servers and reply onions:** A rendezvous server allowed two anonymous circuits to mate. This permitted two parties that were anonymous from both the server and each other to communicate, e.g., in IRC chat. Reply onions built a circuit to connect back to a source that wished to remain anonymous. This could be used for replies to anonymous email or for providing access to a hidden web server. In this way someone who wished to provide a server that could be contacted while remaining anonymous could do so without the risks that affected Penet [29].

Second-generation onion routing:

- **running a client separated from running an onion router:** Perhaps the most important distinction from the first generation. This permitted an increased flexibility in how clients could manage trust and resources. Previously users would be forced to either provide a computer to participate in the onion routing infrastructure themselves or to trust some remote onion router. Now clients could learn about the network and could build their own routes without being required to route traffic for others themselves or trusting a remote computer with control over their routing and knowledge of it.

- **variable length circuits:** The second generation had variable length circuits (up to eleven hops within a single onion, although tunneling to even greater lengths was also possible). In addition to other advantages, this allowed access under different configurations to route within the network in essentially the same way. For example, whether onions were built on a client desktop or at an enclave firewall the rest of the path could be the same.

- **application independent proxies:** A more generic SOCKS proxy was added for those applications that could use it. There was also a redirector that would force all TCP traffic over the onion routing network. This required a kernel shim to be installed, ran only on Windows NT, and (unlike all other onion routing code of all generations) was not built for open source distribution.

- **entry policies and exit policies:** It was recognized that different individuals and organizations running onion routers might have different policy needs. They might want to only permit access to the onion routing network from within an enclave on whose firewall the onion router resided, or only permit exits to that enclave. Alternatively, they might permit email traffic (SMTP) to exit only to some locations and web traffic (HTTP) to exit to arbitrary locations. Thus individual onion routers could set their own entrance and exit policies [50]. This has significant economic impact on network participation and scalability, although the impact of separating clients from onion routers may be even greater [1, 15].

- **dynamic network state:** As in the first-generation, network topology is assumed to be predefined with each node knowing its neighbors. Networks were expected to either be cliques of at most several dozen nodes or, if needed, composed of such cliques with multiple bridges between them. Thus, basic network membership and acceptance of long-term keys were still assumed to be based on outside communication. Nonetheless, link state and other network information were maintained by flooding signed information to the "database engines" (DBE) attached to each onion router. The DBEs were also used to propagate exit policies so that originators would only build circuits to exit the onion routing network at nodes where the intended traffic would be permitted to exit.

- **mixing of cells from different circuits:** Realtime mixing of cells from different anonymous circuits was added. This permitted the exploration of timing, padding, and the like. It was, however, ultimately abandoned in the third-generation until there is at least some theoretical justification for hoping it will provide any benefit against an active adversary.

- **padding and bandwidth limiting:** Traffic is shaped according to a sliding-window weighted average of prior traffic to support protection against a passive eavesdropper doing traffic analysis. Also abandoned in the third-generation.

Third-generation onion routing (Tor):

- **Onion skins not onions: Diffie-Hellman based circuit building:** Perhaps the most important distinction from the first and second generations. As noted above, in the first two generations, circuits were built using an onion structure to distribute session keys. This meant that processed onions had to be kept track of at onion routers until they expired to prevent someone from replaying an onion many times, which might support various kinds of attacks. It also meant that, because session keys were distributed in the onion, anyone who captured an onion and all the (encrypted) traffic on a corresponding circuit could recover all the plaintext if the public keys that encrypted the onion were later broken or otherwise obtained. The onion-based key-distribution protocol was said to lack *forward secrecy*. By extending the circuit one hop at a time, effectively sending a single skin of an onion, session keys could be obtained using a Diffie-Hellman key establishment protocol. This protocol allows the communicants to exchange ephemeral public keys that can be combined with a private key of the other to establish a session key. The session key is never sent, even in encrypted form. This provides forward secrecy, meaning that, even if longterm keys are compromised at some point in the future, session keys will not be. Also, since different ephemeral keys are used at each hop of extending the circuit, there is no threat of replay so there is no need to keep track of onions that have already passed. Interestingly, we considered but abandoned in the spring of 1996 [40] the option of using public Diffie-Hellman values to achieve efficiency gains in computation. Our intended design was to include the public DH-values from the originator inside the layers of circuit building onions, which were used in the first few generations of onion routing designs, and then to combine these with public DH keys (that we assume are DH-values used for generating keys). This is very similar to one of the protocols we introduced much later [42]. Our focus was not on forward secrecy but simply to be more computationally efficient. We were certainly aware of forward secrecy and intentionally chose a protocol for securing links between onion routers that provided it, but we only pursued it with respect to outside attackers rather than against compromised network nodes as well. The idea of using Diffie-Hellman for basic circuit building was simply another dropped design idea until work began on the Tor design, when it was picked up for the forward secrecy it provided and for freedom from the need to store onions against replay. The first description [27, 43] and implementation of onion routing uses RSA public keys for distributing circuit session keys and DH-established link encryption between the server nodes. The current version of onion routing, Tor, uses both a Diffie-Hellman key exchange and an RSA encryption/decryption for each step on the anonymizing tunnel setup. The computational advantages of using Diffie-Hellman that we contemplated in 1996 lay dormant until 2007, when we picked it up again, as did others [42, 34, 33].

- **Fixed-length three-hop circuits:** In Tor, primary emphasis is placed on what was dubbed in the second-generation, the "customer-ISP configuration" [44, 50] or more generally, "remote-COR configuration" [51], meaning that circuits are built by a local client and enter the onion routing network at some remote onion router after passing over some publicly visible network. This, coupled with an abandonment of any pursuit of mixing, reduces motivation for more than three hops to a circuit. And three hops is minimal to protect against two concerns. The first concern is that either the entry or exit node for the onion routing network might be sensitive, e.g., if it is on an enclave firewall. In this case two-hop routes have a single point of failure to link a sensitive source to a destination or vice versa. The second concern is that, in general, with two-hop circuits a compromised entry or exit would immediately know for each connection through it the single other point to attack to reveal the entire route. If the adversary has resources that can be readily mobilized for attacking at some of the nodes in the network when needed, two-hop circuits would make his job much easier than three-hop circuits, for which he would need to simply be lucky in knowing where to strike and when, or would need to keep his resources persistently mobilized everywhere.

- **Rendezvous circuits and hidden servers:** First-generation onion routing introduced rendezvous servers, for example to permit people to onion route to a chat server. And, hidden services were to be accessed via reply onions. Tor did away with onions per se and thus introduced a design for hidden services based on rendezvous circuits. On the one hand, this meant that one did not need to worry if the onion routers in a reply onion would be functional at the time the reply onion was actually used. On the other hand, one had to maintain open circuits to introduction locations at which the hidden server could be contacted. (Actual data is passed over a separate circuit to a rendezvous point in order to minimize the traffic load and the responsibility of the introduction points.)

- **Directory servers:** First-generation onion routing simply assumed network information was static or distributed offline. Second-generation onion routing assumed basic network membership information to be established offline but used a flooding mechanism to distribute authenticated network link state, keying information, and policy information necessary to build circuits. Third-generation onion routing introduced directory servers to distribute network status, but also network membership information. This is less complex, more flexible, and more scalable than flooding for maintaining a consistent picture of the network. On the other hand, this change requires a handful of trusted directory servers. Caching at other onion routers reduces the risk of a performance bottleneck, but this remains a trust bottleneck. Since the original design, Tor has evolved through several versions of directory structure to reduce the trust placed in individual directory servers, to increase resilience of the network to faulty or malicious directory servers, and to reduce overhead.

- **most application proxies eliminated as unnecessary:** SOCKS is a protocol originally designed to permit applications to communicate across firewalls in a flexible and controlled manner. A SOCKS proxy was added to second-generation onion routing, but other proxies were still needed because adoption of SOCKS was still limited. By the third generation, SOCKS had become common enough that applications could be assumed to use it and specific application proxies were no longer needed. Nonetheless, some issues remained. For example the most advanced version of SOCKS did not proxy DNS requests, only an intermediate version did. In addition, much network identification information was often passed by applications regardless of anonymization of the connection. For this reason, the first two generations of onion routing included sanitizing proxies to reduce this. By the time of third generation, sanitizing proxies were being developed and made freely available by others. Thus onion routing could focus on its primary job of anonymizing the circuits and use what others provided if application sanitization were needed. Note that sanitization should be kept a separate function. (If one were logging into work via SSH or a VPN over onion routing from a remote location, for example, then such sanitization is unnecessary and might hamper some authentication or functionality.) As in the directory system, Tor itself has evolved since 2004. In addition to SOCKS support as above, it now allows the use of transparent proxies (similar in function to the second-generation redirector for Windows NT), and it can process DNS queries as if a DNS server itself.

The very first onion routing design was peer-to-peer in the literal sense that all system elements interacted as peers. But it was not a P2P design in the sense that all end-user computers were expected to contribute to the infrastructure. P2P onion routing designs in that sense have also received a lot of attention. I simply cite a sampling of these designs without analysis [45, 24, 39, 36, 35]. So far, every P2P onion routing system has been shown within a year or so of publication to have one or another significant break. Having a completely decentralized design for node discovery and/or route propagation, whether or not the lookup is structured, is complex. I expect research in this area to remain both interesting and volatile for a while, but we shall see.

Both second and third generation onion routing have a client-server architecture, but where the server nodes are entirely locally managed and network membership and status is itself distributed. The second generation design for this was via a flat propagation of network information (that was never fully implemented and deployed). The current public Tor network has on the order of ten directory authorities that are independently managed and exist in multiple locations and jurisdictions. We have already mentioned this above, but here we note that having such an architecture can allow for significant scaling that is often cited as a goal of P2P design. Allowing clients to participate without having to provide servers means that many more can participate. Allowing this while still having a large diverse network provides performance and protection that has yet to be achieved by P2P designs [13].

We have not covered all of the features and differences of the three generations on onion routing designed at NRL, but we have hopefully covered the most salient basic distinctions. Distinguishing the names of the different designs also requires some explanation. The title of the Tor design paper as originally published is "Tor: The Second-Generation Onion Router"[17]. This was a misleading title in at least three respects, which we will now set out. We will also explain the origins of the name 'Tor'. To some extent, getting all this straight will help readers have a clearer understanding of onion routing and its history.

When the second-generation design above was begun, it was called "Onion Routing, The Next Generation" out of homage to a certain television series that was popular at the time, particularly with some of the onioneers. It was thus not only a second generation of onion routing, but one that was deliberately considered another generation by its designers. By contrast, when work on the Tor design began second-generation prototypes had been languishing for a few years without much attention. Tor began in a project originally intended to simply clean up and update the implementation of the second-generation software and tweak the design where needed for better performance and fault tolerance. However, it was soon recognized that a more radical redesign and re-implementation would more quickly lead to a better system overall. One of Tor's designers tended to use "first-generation" for all work on onion routing prior to when he started. Thus to him, Tor was second-generation. One of the designer's should have known better because he was there for all the history of all the generations. However, at the time he was inclined to not care about titles and go along with whatever his co-authors suggested for a title, not realizing that it would make describing the various systems more complicated in the future. Thus, though structurally Tor is clearly a third generation of basic design, the title of this paper would lead one to think otherwise.

The title is also confusing because it refers to a second-generation onion router, that is an onion-routing network component through which circuits are built and that forwards traffic along those circuits. But the paper is also about the onion-routing clients, the hidden service design, the directory system, etc., in other words an entire onion routing system design. Though onion routers are a central part of onion routing, describing them is a small part of the paper. In addition, by phrasing the title "Tor: The Second-Generation Onion Router", we gave an at least implicit impression that 'Tor' stood for 'the onion router'. This further exacerbated the confusion about the paper's scope and focus, but it is also simply incorrect.

When Roger Dingledine began work on onion routing as part of an NRL project, there were many other onion routing systems that had been or were being designed, published, and/or deployed, for example, Freedom [26], Cebolla [8], and Tarzan [24]. Thus, when he told people he was working on onion routing, they would ask him which one. He would respond that it was *the* onion routing, the original program of projects from NRL. It was Rachel Greenstadt who noted to him that this was a nice acronym and gave Tor its name. Roger then observed that it also works well as a recursive acronym, 'Tor's onion routing'. It was also his decision that it should be written 'Tor' not 'TOR'. Making it more of an ordinary word in this way also emphasizes the overlap of meaning with the German word 'Tor', which is gate (as in a city gate).

To sum up, "Tor: The Second-Generation Onion Router" is about the design of onion-routing systems, not just onion routers themselves. Tor is the third generation of onion routing, not the second. And the 'r' in 'Tor' represents 'routing' not 'router'. In hindsight we probably should have spent a bit more time on the paper title.

4.1 Freedom's just another word for no peel left to loose

As noted, within a short time after onion routing was invented, there were several onion routing designs developed elsewhere than at NRL. A significant contribution that fed back into NRL onion routing design evolution was Zach Brown's Cebolla [8]. As also noted above, perhaps the most significant discriminator of the third generation NRL onion routing design from second generation was the introduction of incremental circuit building via a Diffie-Hellman exchange between each node in the route and the client over the existing partially built circuit. The idea for this came from Cebolla.

The most notable onion routing network prior to Tor in terms of the scale of its development and deployment was the Freedom Network from Zero-Knowledge Systems, Inc. It was publicly announced in late 1998, and a deployed network ran from late 1999 till late 2001. This was a commercial system in which users paid for service. That itself is not without security implications, which we have already touched on briefly. (This is not the only commercial onion routing network that has existed. For example, IronKey has sold protected flash drives since about 2006, including one that supports private web browsing through IronKey's own onion routing network.) We focus here on Freedom's network design and protocols. Like the first two generations of onion routing, the first version of Freedom built a route using an onion that passed the session keys to each of the onion routers—called Anonymous Internet Proxies (AIP) in the Freedom Network [26]. The resulting encryption-layered route was dubbed a "telescope" by the Freedom designers, after the nested tube structure of spyglass telescopes.

But Freedom had several important differences from the NRL designs for onion routing. It transported traffic using UDP rather than TCP. It used block ciphers since, unlike NRL onion routing, packets could not be guaranteed to be in-order. However, since many of the applications it carried required TCP guarantees, the Freedom client managed its own TCP stack (similarly for network exit nodes). This required a kernel modification and thus root access on the computer on which the program ran. A resulting tradeoff was that this gave Freedom tighter control over making sure that applications did not bypass the anonymity network than any of the NRL versions of onion routing except perhaps the redirector mentioned above. Freedom also provided a deployed pseudonymous mail infrastructure, including standard features such as spam control but also *nymservers* that would allow one to look up a pseudonym and obtain a reply block (essentially like the reply onions described above) and send reply traffic through the Freedom Network to the appropriate pseudonymous recipient. While the first two generations of NRL onion routing supported SMTP for sending mail and some sort of integration with reply onions was envisioned, none was ever fully designed much less deployed.

Probably the most important difference between NRL-design onion routing and Freedom was the pseudonym system. This was not simply to enable replies to email from Freedom clients, it pervaded the Freedom architecture. The underlying communications architecture was still an onion routing system that separated identification from routing and was thus an anonymity system in that respect, but the pseudonym system that ran on top of it was central to the users' interactions with the whole system. As the designers noted, you should "think of Freedom as a pseudonymity product, rather than as an anonymity product" [25]. Users were "encouraged to create pseudonyms ('nyms') for each area of activity in which they want to preserve their privacy" [7]. Nyms were purchased in a process that prevented linking of nyms to a given client. Freedom circuits connecting clients to Internet destinations were authenticated at the exit node via a pseudonymous signature from the client. Any exit node would be able to link two connections made by a client using the same nym no matter when those connections were made. Thus, though the network and routing structure of Freedom onion routing and NRL-design onion routing are very similar, the goals are somewhat different. NRL-design onion routing does not have pseudonyms, but all traffic that passes over a single circuit can be linked together as coming from the same unknown client. First generation NRL onion routing built a new circuit for each HTTP request, which was in keeping with the way HTTP 1.0 worked. The circuit-building overhead of that approach is, however, quite high. Second and third generation onion routing would thus multiplex several application connections over a single circuit. By default Tor reuses a circuit for ten minutes, although a GUI button allows users to rotate to a new circuit at any time. Still an exit node or collaborating exit nodes in the Freedom Network seeing the same pseudonym at any time, would be able to link these together. Freedom was designed to prevent system components from linking a pseudonym to its client, not against pseudonymous profiling by the exit node or other properties. To the external destination, however, Freedom did not reveal nyms and thus provided client anonymity. But, it did not hide HTTP referer fields and thus permitted that degree of linking even across nyms. If this was a concern, it was recommended that in the course of your browsing session you return in between to a homepage that was a popular web page [25] or simply blank [3].

In the Freedom 2.0 architecture [7] mail reply blocks were replaced with a different design that was intended to improve capacity, performance, and security. It was, in too tight a nutshell, a bulletin board system that allowed people outside the Freedom system to post messages to a nym at a POP server. These could then be retrieved by Freedom clients via anonymous circuits through the network. Another important change to Freedom in the 2.0 architecture was to change to two hops for circuits to the Internet [2]. In Freedom 2.1, this was further reduced to one hop as a default, with a user setable option for two or three hops depending on the desired security/performance tradeoff [3]. This has two security implications. First, anonymity is reduced to a single point of failure, albeit a less predictable one in a distributed network like Freedom's. Also, in the case of Freedom, it is not just the dissociation of the source and destination(s) of that particular circuit that is lost; the association of a nym with a client IP address is also revealed to a compromised onion router. Second, letting users select which connections should have more hops automatically partitions the usage anonymity set, and it indicates to the exit node whether the connection is considered sensitive by the user, hence deserving of closer scrutiny. The designers were aware of various security implications of these choices but also recognized the potential for performance improvements if the number of circuit hops was reduced. These issues were discussed in the above-cited papers.

5. WHAT ONION ROUTING

Onion routing separates identification from routing. But can we describe what protection that provides or tell how strong that protection is? When the first onion routing paper was submitted, it was taken by at least some reviewers as an anonymous communications protocol akin to Chaum mixes and providing the same sort of security. Anonymity (security of anonymous communication) has since at least that time been measured by the size and sometimes distribution on an anonymity set. For protecting a communications initiator, this would thus reflect the uncertainty within a given set of who initiated that communication. The other primary ingredient in defining and measuring security besides the property being protected is the adversary attempting to undermine that security. For much of the history of anonymous communication, this has been the global passive adversary (GPA), one that can observe all messages everywhere, but cannot alter or delete messages or insert its own messages. Onion routing is completely broken against a GPA. It has long been documented that an adversary observing both ends of a circuit can trivially associate those ends by means of timing correlation or volume [46, 41]. In fact, the circuit does not even need to carry any traffic; circuit setup is enough [5]. To date, no padding or similar scheme has been devised that will counter any sort of realistic adversary (not GPA, but we'll return to that momentarily) while preserving anything like the latency properties expected for most applications on onion routing networks. Work on this remains interesting and worth pursuing [23], but I am dubious that any practical solution exists.

Mixes, on the other hand, break up communication patterns for messages by introducing delays and reordering. They are not trivially broken against a GPA. For this reason, it is common to characterize onion routing as less secure than mixing, if more practical and thus a good engineering tradeoff. But this is too simple. There are realistic adversaries and realistic settings in which an onion routing network is more secure than the comparable mix network that is essentially the same except for swapping mixes for onion routers at the network nodes [49]. Another problem with a GPA that we have described since the early days of onion routing is that a GPA is both too strong and too weak. It is too strong because it is unlikely that an adversary can observe all activity at all places of a large global network. The adversary may be able to observe a large fraction, but not all traffic, everywhere, at all times. This could be viewed as a conservative adversary, as strong or stronger than anything one will face in practice. But it is also too weak because it cannot even delay packets anywhere for a nanosecond or interact with the system by acting as an ordinary user. We can strengthen the adversary by making it less passive, able to use the system as an ordinary user and to own some network nodes at which it can do anything computationally feasible with traffic at that node. This is done in some work on both mix networks [12] and onion routing networks [51].

More importantly, however, this whole approach to security misconstrues both adversary model and definition of security in a way that leads both design and analysis in the wrong direction. The entropist approach [48] assumes that an adversary has a known set of entities (for example, senders if we wish to protect the identity of senders) and that its goal is to reduce its uncertainty about the sender on that set.

A good measure of security is the amount of work an adversary must do to break it. If reducing the size of the anonymity set or improving the chance of guessing the right sender within it is indicative of the work the adversary must do, then this is a good measure. And this is indeed true for systems such as those for voting anonymity, where the goal is to prevent adversaries from associating individuals from a known set of registered voters with the votes cast. But that is not the case for large public onion routing networks. For example, if I am using Tor and both ends of my circuit go through adversary controlled or observed onion routers, it will not matter if there are five other users of the Tor network then or five hundred thousand. And it will not matter how many of those other users have circuits running through the same terminal onion routers or not. An adversary might make use of such numbers. For example, if he could determine that the network only has a small enough number of users that it is practical to just identify and attack them directly to observe their future behavior, then this would be useful. But directly determining a set and reducing uncertainty within it would be a waste of effort. (I am speaking generally. There are always exceptional circumstances.) As noted, a realistic typical attacker will not actually know or care about this when breaking anonymity on an onion routing system. Perhaps we are looking at a set of the wrong things. An adversary wins if he owns the onion routers at each end. So perhaps he should care about the number of onion routers and reducing uncertainty on that set.

First, there are attacks that identify terminal onion routers themselves for some network sizes and configurations [37]. But, outside of P2P designs, that does not connect the source client to the destination. It could help determine where to direct further attacks, but reducing uncertainty on that set is not itself a good measure of anonymity of system users.

Where to direct further attack does matter. This is especially true in the Tor network since 2006, since the deployment of guard nodes. Since that time, instead of building a circuit through three randomly chosen onion routers, the first hop is chosen from a small set (default size three) of guard nodes that are used by a client as long as they are available. Why are there guard nodes? We showed in 2005 that an attacker owning a single onion router could find a hidden server in a matter of minutes by repeatedly making connections to it until the hidden server's rendezvous circuit connected through the attacker's onion router [41]. (Hidden servers are briefly described above in section 4.) Because we wanted to show what could be accomplished owning a single node in the network, the attack only worked on hidden services. We noted that if the adversary owned two or more onion routers, similar attacks could be carried out on ordinary Tor connections to public servers. This was later empirically demonstrated [5]. These attacks were already known, and guard nodes are a version of the earlier introduced concept of "helper nodes" which were meant to help resist such attacks for a variety of anonymous communications protocols, not just onion routing. Though they had been shown possible, our 2005 analysis of hidden services was surprising in showing how cheap and efficient they were: a single adversary node could find a hidden server in just a few minutes. Thus guards were deployed as a countermeasure.

Guard nodes resist attacks that watch for or cause repeated circuit formation until an adversary node is chosen for the first onion router adjacent to the client. Introducing them also meant that an adversary that identifies a guard node, e.g., of someone repeatedly visiting a particular website, knows that if he attacks it he will be able to see one end of a large fraction of the target client's connections. He still needs to be able to watch the other end to take advantage of that. If he owns a destination website and wants to know who visits it, that part will already be accomplished.

This brings us back to defining and measuring security for onion routing. The measure of security is the amount of work an adversary must do to break it. If all nodes in the network are equally vulnerable to compromise by an adversary, then the number of nodes serves as a good indicator of the work an adversary must do to attack onion routing as we have been describing. And this brings us to another mistake of entropism and the designs it engenders. The number of nodes in the network is only a good indicator of the work an adversary must do if they *are* generally comparable in their vulnerability to compromise. But both designs and their analyses treat all parts of the network the same. Tor is a volunteer network. This has much to do with its growth and viability to the largest by far network for privacy and traffic analysis resistance [1, 15] and continues a trend begun in the first generation design with open source of code and plans for a network controlled by multiple not necessarily mutually-trusting but mutually-collaborating node providers, and then continued in second generation's introduction of allowing node operators to choose their own individual exit policies, etc. But along with this comes an obvious diversity of how likely an adversary is to desire or be able to own or observe network nodes. That sort of diversity is there for any viable large open network; it should just be very obvious with Tor. And if the adversary has a botnet, he could easily, gradually, and stealthily add his own nodes to the network in wide portions of the geographic and IP space until he owns a significant fraction of the network and can observe much of the traffic. This means that the number of nodes tells us little about how resistant onion routing is to an adversary with more than trivial resources and skill level.

To deal with botnets or more generally this diversity of trust in various nodes in the network, we could just stick to those nodes that we trust, for example, that we run ourselves. This returns us to the original reason that onion routing was not deployed as a Navy-only or similarly restricted system. Nodes that are trusted by us may either be known by external indicators and/or identified through behavior. This would undermine the protection that onion routing is meant to provide. So we need a more subtle way to reason about trust. In our current work, we identify trust in a node with the complement of the probability that the node is compromised. We have shown some optimality results for choosing first and last nodes in an onion routing circuit against an adversary trying to own both of them [30]. We also have devised and analyzed algorithms for routing securely even when an adversary might own large portions of the network and when we might be somewhat inaccurate in assigning trust to nodes in the network [31]. We intend to continue to develop this area in various ways, for example, by incorporating link threats into the model [22, 38, 20].

5.1 Traffic Security and Identity

We must overcome one more vestige of the entropist view of anonymity: the notion that the paramount thing to protect is the identification of a specific individual. We have already noted that the standard (entropist) characterization of anonymous communications security starts with a given set of distinct users, messages, source IP addresses, etc. The goal of a security system is then presented as preventing identification of the unique one that matches a given communication elsewhere. This communication elsewhere might be a given message on the other side of a mix or a query or post at a website. More generally we might wish to merely resist reduction of uncertainty about which one is the target individual or about the overall matching of multiple communications with multiple targets. Often, however, this is simply not what matters. If I am a patent examiner wishing to hide that I am searching for prior art on someone's website, I am not trying to prevent him from knowing which patent examiner or source computer is visiting his site. I want to protect that a patent examiner is visiting his site. Similarly if I am using the Internet while traveling and wish to protect my U.S. Government affiliation from local network observers, I am primarily concerned with that. I may not care if he learns specific IP addresses I contact. What I do not want him to do is recognize my affiliation. He could do this if, e.g., a large fraction of my traffic appears most likely headed to .mil or .gov parts of IP space. He may not know any specific IP addresses, and that may not matter to him or be of secondary importance. It is true that an adversary who can disambiguate individual users or source addresses, and who has the appropriate ancillary information, might use this to help him achieve his primary goal. But it is not itself always his primary goal. Sometimes it is, but not always. The entropist view tries to fit all anonymous communication into this procrustean metric.

If protecting identity is sometimes about hiding association with a property that is possibly shared with many others, what about when a property to be protected is itself that you are trying to obtain such protection? Onion routing can hide your ultimate destination from a local observer, but as described so far it does not hide that you are using an onion routing network. The relays in the Tor network are publicly listed in directories. This allows Tor clients to gather the information needed to build circuits. It also means that someone who runs a website and does not want to allow access to it from the Tor network has an easy means to do so. There is even a script made publicly available by the Tor Project to facilitate this. Blocking typically does not have the intended effect; it usually only prevents access by the honest because the dishonest have many less salient ways of access available to them. For this reason blocking is discouraged; however, if someone wants to block access to his server from Tor, he can do so. But this also means that one can block access *to* the Tor network.

Many people use Tor to access search engines, such as Google, so that they can see the results that one would get in a different location. (Search results are in part determined by the IP address of a request. Tor user's are sometimes surprised the first time a search engine homepage loads for them in a different language.) And some search engines, media outlets, or other destinations are blocked or filtered by national or other jurisdictional firewalls. If people use Tor to obtain unfettered access to the Internet, the adversary may block access to the public Tor network itself. Separating identification from routing may then need to be applied to itself. How can we separate identification as onion routing communication from routing to and through an onion routing network?

The Tor network uses *bridges.* These are onion routers that provide access to the rest of the Tor network but are not themselves publicly listed. A major goal is to help unknown individuals behind such firewalls and filters to access the public Tor network in this way—which creates a new problem. How do you tell these people about the bridges without also telling those who want to block them and thus letting the bridges get blocked too? There are various strategies that we only have space to touch on here, such as trickling out the bridge addresses through a limited resource (such as to email requests from accounts with an email provider that does not make it trivial to sign up for numerous accounts), or rapidly rotating bridge addresses through a portion of IP space that an adversary does not want to permanently leave blocked [16, 14]. It also becomes important that traffic to and from the Tor network, even via bridges, is not trivially separable by some aspect of the connection protocol or of the communication patterns. An example of very recent work on blocking resistance via Tor bridges is BridgeSPA [47].

6. CONCLUSION

Why, you old soothsayer-humbug!
no Kaiser are you; you are nought but an onion.
I'm going to peel you now, my good Peer!
You won't escape either by begging or howling.
. . .
What an enormous number of swathings!
Isn't the kernel soon coming to light?
. . .
I'm blest if it is! To the innermost centre,
it's nothing but swathings-each smaller and smaller.
Nature is witty!
. . .
A queer enough business, the whole concern!
Life, as they say, plays with cards up its sleeve;
but when one snatches at them, they've disappeared,
and one grips something else, or else nothing at all.

—Henrik Ibsen, *Peer Gynt*, Act. V, Scene 5

In this paper, we have discussed some of the why, who, how, which, and what of onion routing (also some of the when, although that was scattered throughout). Onion routing has grown as both a research area and an applied solution to many real problems over the last decade and a half. We have barely scratched the outermost layer of many of its aspects in this paper and not touched others at all. If you now have the motivation to explore more deeply, I hope that unlike Peer Gynt in his introspection, you will find something of substance in the process.

7. REFERENCES

[1] Alessandro Acquisti, Roger Dingledine, and Paul Syverson. On the economics of anonymity. In Rebecca N. Wright, editor, *Financial Cryptography, 7th International Conference, FC 2003*, pages 84–102. Springer-Verlag, LNCS 2742, 2003.

[2] Adam Back, Ian Goldberg, and Adam Shostack. Freedom systems 2.0 security issues and analysis. White paper, Zero Knowledge Systems, Inc., October 2001. The attributed date is that printed at the head of the paper. The cited work is, however, superceded by documents that came before Oct. 2001, e.g., [3].

[3] Adam Back, Ian Goldberg, and Adam Shostack. Freedom systems 2.1 security issues and analysis. White paper, Zero Knowledge Systems, Inc., May 2001.

[4] Adam Back, Ulf Möller, and Anton Stiglic. Traffic analysis attacks and trade-offs in anonymity providing systems. In Ira S. Moskowitz, editor, *Information Hiding: 4th International Workshop, IH 2001*, pages 245–257, Pittsburgh, PA, USA, April 2001. Springer-Verlag, LNCS 2137.

[5] Kevin Bauer, Damon McCoy, Dirk Grunwald, Tadayoshi Kohno, and Douglas Sicker. Low-resource routing attacks against Tor. In Ting Yu, editor, *WPES'07: Proceedings of the 2007 ACM Workshop on Privacy in the Electronic Society*, pages 11–20. ACM Press, October 2007.

[6] Oliver Berthold, Hannes Federrath, and Stefan Köpsell. Web MIXes: A system for anonymous and unobservable Internet access. In Hannes Federrath, editor, *Designing Privacy Enhancing Technologies: International Workshop on Design Issues in Anonymity and Unobservability*, pages 115–129. Springer-Verlag, LNCS 2009, July 2000.

[7] Philippe Boucher, Adam Shostack, and Ian Goldberg. Freedom systems 2.0 architecture. White paper, Zero Knowledge Systems, Inc., December 2000.

[8] Zach Brown. Cebolla: Pragmatic IP anonymity. In *Proceedings of the 2002 Ottawa Linux Symposium*, June 2002.

[9] David Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Communications of the ACM*, 4(2):84–88, February 1981.

[10] Wei Dai. Pipenet, February 1995. Original suggestion posted to the cypherpunks mailing list in Feb. 1995. Later versions are in the Free Haven Anonymity Bibliography.

[11] George Danezis, Claudia Diaz, and Paul Syverson. Anonymous communication. In Burton Rosenberg, editor, *Handbook of Financial Cryptography.* CRC Press, 2010.

[12] George Danezis, Roger Dingledine, and Nick Mathewson. Mixminion: Design of a type III anonymous remailer protocol. In *Proceedings, 2003 IEEE Symposium on Security and Privacy*, pages 2–15, Berkeley, CA, May 2003. IEEE Computer Society.

[13] George Danezis and Paul Syverson. Bridging and fingerprinting: Epistemic attacks on route selection. In Nikita Borisov and Ian Goldberg, editors, *Privacy Enhancing Technologies: Eighth International Symposium, PETS 2008*, pages 151–166. Springer-Verlag, LNCS 5134, July 2008.

[14] Roger Dingledine. Strategies for getting more bridge addresses. `https://blog.torproject.org/blog/strategies-getting-more-bridge-addresses`, May 2011.

[15] Roger Dingledine and Nick Mathewson. Anonymity loves company: Usability and the network effect. In Ross Anderson, editor, *Fifth Workshop on the Economics of Information Security (WEIS 2006)*, June 2006.

[16] Roger Dingledine and Nick Mathewson. Design of a blocking-resistant anonymity system (draft). `https://svn.torproject.org/svn/projects/design-paper/blocking.html`, November 2006.

[17] Roger Dingledine, Nick Mathewson, and Paul Syverson. Tor: The second-generation onion router. In *Proceedings of the 13th USENIX Security Symposium*, pages 303–319. USENIX Association, August 2004.

[18] Roger Dingledine, Nick Mathewson, and Paul Syverson. Challenges in deploying low-latency anonymity (draft). NRL CHACS Report 5540-625, 2005.

[19] Roger Dingledine, Nick Mathewson, and Paul Syverson. Deploying low-latency anonymity: Design challenges and social factors. *IEEE Security & Privacy*, 5(5):83–87, September/October 2007.

[20] Matthew Edman and Paul Syverson. AS-awareness in Tor path selection. In Somesh Jha, Angelos D. Keromytis, and Hao Chen, editors, *CCS'09:*

Proceedings of the 16th ACM Conference on Computer and Communications Security, pages 380–389. ACM Press, 2009.

[21] Matthew Edman and Bülent Yener. On anonymity in an electronic society: A survey of anonymous communication systems. *ACM Computing Surveys*, 42(1), 2010.

[22] Nick Feamster and Roger Dingledine. Location diversity in anonymity networks. In Sabrina De Capitani di Vimercati and Paul Syverson, editors, *WPES'04: Proceedings of the 2004 ACM Workshop on Privacy in the Electronic Society*, pages 66–76, Washington, DC, USA, October 2004. ACM Press.

[23] Joan Feigenbaum, Aaron Johnson, and Paul Syverson. Preventing active timing attacks in low-latency anonymous communication [extended abstract]. In Mikhail J. Attallah and Nicholas J. Hopper, editors, *Privacy Enhancing Technologies: 10th International Symposium, PETS 2010*, pages 166–183. Springer-Verlag, LNCS 2605, July 2010.

[24] Michael J. Freedman and Robert Morris. Tarzan: A peer-to-peer anonymizing network layer. In Vijay Atluri, editor, *Proceedings of the 9th ACM Conference on Computer and Communications Security, CCS 2002*, pages 193–206. ACM Press, 2002.

[25] Ian Goldberg and Adam Shostack. Freedom 1.0 security issues and analysis. White paper, Zero Knowledge Systems, Inc., November 1999.

[26] Ian Goldberg and Adam Shostack. Freedom network 1.0 architecture and protocols. White paper, Zero Knowledge Systems, Inc., October 2001. The attributed date is that printed at the head of the paper. The cited work is, however, superceded by documents that came before Oct. 2001. The appendix indicates a change history with changes last made November 29, 1999. Also, in [25] the same authors refer to a paper with a similar title as an "April 1999 whitepaper".

[27] David M. Goldschlag, Michael G. Reed, and Paul F. Syverson. Hiding routing information. In Ross Anderson, editor, *Information Hiding: First International Workshop*, pages 137–150. Springer-Verlag, LNCS 1174, May 1996.

[28] Ceki Gülcü and Gene Tsudik. Mixing E-mail with Babel. In *Proceedings of the Symposium on Network and Distributed Security Symposium - NDSS '96*, pages 2–16. IEEE, February 1996.

[29] Sabine Helmers. A brief history of anon.penet.fi - the legendary anonymous remailer. *CMC Magazine*, September 1997.

[30] Aaron Johnson and Paul Syverson. More anonymous onion routing through trust. In *22nd IEEE Computer Security Foundations Symposium, CSF 2009*, pages 3–12, Port Jefferson, New York, USA, July 2009. IEEE Computer Society.

[31] Aaron Johnson, Paul Syverson, Roger Dingledine, and Nick Mathewson. Trust-based anonymous communication: Adversary models and routing algorithms. In George Danezis, Vitaly Shmatikov, and Dongyan Xu, editors, *CCS'11: Proceedings of the 18th ACM Conference on Computer and Communications Security*. ACM Press, 2011.

[32] JonDonym – the internet anonymisation service. `https://www.jondos.de/en/`, 2008. Commercial version of the Java Anon Proxy (JAP). Initially published description in [6].

[33] Aniket Kate and Ian Goldberg. Using Sphinx to improve onion routing circuit construction (extended abstract). In Radu Sion, editor, *Financial Cryptography and Data Security, 14th International Conference, FC 2010, Revised Selected Papers*, pages 359–366. Springer-Verlag, LNCS 6052, 2010.

[34] Aniket Kate, Greg Zaverucha, and Ian Goldberg. Pairing-based onion routing. In Nikita Borisov and Philippe Golle, editors, *Privacy Enhancing Technologies: 7th International Symposium, PET 2007*, pages 95–112. Springer-Verlag, LNCS 4776, 2007.

[35] Jon McLachlan, Andrew Tran, Nicholas Hopper, and Yongdae Kim. Scalable onion routing with torsk. In Somesh Jha, Angelos D. Keromytis, and Hao Chen, editors, *CCS'09: Proceedings of the 16th ACM Conference on Computer and Communications Security*, pages 590–599. ACM Press, 2009.

[36] Prateek Mittal and Nikita Borisov. Shadowwalker: Peer-to-peer anonymous communication using redundant structured topologies. In Somesh Jha, Angelos D. Keromytis, and Hao Chen, editors, *CCS'09: Proceedings of the 16th ACM Conference on Computer and Communications Security*, pages 161–172. ACM Press, 2009.

[37] Steven J. Murdoch and George Danezis. Low-cost traffic analysis of Tor. In *2005 IEEE Symposium on Security and Privacy,(IEEE S&P 2005) Proceedings*, pages 183–195. IEEE CS, May 2005.

[38] Steven J. Murdoch and Piotr Zieliński. Sampled traffic analysis by internet-exchange-level adversaries. In Nikita Borisov and Philippe Golle, editors, *Privacy Enhancing Technologies: 7th International Symposium, PET 2007*, pages 167–183. Springer-Verlag, LNCS 4776, 2007.

[39] Arjun Nambiar and Matthew Wright. Salsa: A structured approach to large-scale anonymity. In Rebecca N. Wright, Sabrina De Capitani di Vimercati, and Vitaly Shmatikov, editors, *CCS'06: Proceedings of the 13th ACM Conference on Computer and Communications Security*, pages 17–26. ACM Press, 2006.

[40] Onion routing: Brief selected history. `http://www.onion-router.net/History.html`, 2006.

[41] Lasse Øverlier and Paul Syverson. Locating hidden servers. In *2006 IEEE Symposium on Security and Privacy (S& P 2006), Proceedings*, pages 100–114. IEEE CS, May 2006.

[42] Lasse Øverlier and Paul Syverson. Improving efficiency and simplicty of Tor circuit establishment and hidden services. In Nikita Borisov and Philippe Golle, editors, *Privacy Enhancing Technologies: 7th International Symposium, PET 2007*, pages 134–152. Springer-Verlag, LNCS 4776, 2007.

[43] Michael G. Reed, Paul F. Syverson, and David M. Goldschlag. Proxies for anonymous routing. In *Twelfth Annual Computer Security Applications Conference*, pages 95–104. IEEE CS Press, 1996.

[44] Michael G. Reed, Paul F. Syverson, and David M. Goldschlag. Anonymous connections and onion routing. *IEEE Journal on Selected Areas in Communications*, 16(4):482–494, May 1998.

[45] Marc Rennhard and Bernhard Plattner. Introducing MorphMix: Peer-to-peer based anonymous internet usage with collusion detection. In Sabrina De Capitani di Vimercati and Pierangela Samarati, editors, *Proceedings of the ACM Workshop on Privacy in the Electronic Society, WPES 2002*, pages 91–102. ACM Press, 2002.

[46] Andrei Serjantov and Peter Sewell. Passive attack analysis for connection-based anonymity systems. In Einar Snekkenes and Dieter Gollmann, editors, *Computer Security – ESORICS 2003, 8th European Symposium on Research in Computer Security*, pages 141–159, Gjøvik, Norway, October 2003. Springer-Verlag, LNCS 2808.

[47] Rob Smits, Divam Jain, Sarah Pidcock, Ian Goldberg, and Urs Hengartner. Bridgespa: Improving tor bridges with single packet authorization. In Jaideep Vaidya, editor, *WPES'11: Proceedings of the 2011 ACM Workshop on Privacy in the Electronic Society*. ACM Press, October 2011.

[48] Paul Syverson. Why I'm not an entropist. In *Seventeenth International Workshop on Security Protocols*. Springer-Verlag, LNCS, 2009. Forthcoming.

[49] Paul Syverson. Sleeping dogs lie in a bed of onions but wake when mixed. In *4th Hot Topics in Privacy Enhancing Technologies (HotPETs 2011)*, July 2011.

[50] Paul Syverson, Michael Reed, and David Goldschlag. Onion Routing access configurations. In *Proceedings DARPA Information Survivability Conference & Exposition, DISCEX'00*, volume 1, pages 34–40. IEEE CS Press, 1999.

[51] Paul Syverson, Gene Tsudik, Michael Reed, and Carl Landwehr. Towards an analysis of onion routing security. In Hannes Federrath, editor, *Designing Privacy Enhancing Technologies: International Workshop on Design Issues in Anonymity and Unobservability*, pages 96–114. Springer-Verlag, LNCS 2009, July 2000.

[52] Paul F. Syverson, David M. Goldschlag, and Michael G. Reed. Anonymous connections and onion routing. In *Proceedings, 1997 IEEE Symposium on Security and Privacy*, pages 44–54. IEEE CS Press, May 1997.

[53] Tor Metrics Portal. `https://metrics.torproject.org/`, 2011.

The Science of Cyber Security Experimentation: The DETER Project

Terry Benzel
USC Information Sciences Institute
4676 Admiralty Way #1001
Marina del Rey, CA 90292
+1-310-822-1511
tbenzel@isi.edu

ABSTRACT

Since 2004, the DETER Cyber-security Project has worked to create an evolving infrastructure – facilities, tools, and processes – to provide a national resource for experimentation in cyber security. Building on our insights into requirements for cyber science and on lessons learned through 8 years of operation, we have made several transformative advances towards creating the next generation of DeterLab. These advances in experiment design and research methodology are yielding progressive improvements not only in experiment scale, complexity, diversity, and repeatability, but also in the ability of researchers to leverage prior experimental efforts of other researchers in the DeterLab user community. This paper describes the advances resulting in a new experimentation *science* and a transformed facility for cyber-security research development and evaluation.

Categories and Subject Descriptors

D.4.6 Security and Protection D.4.8 Performance Measurements - Modeling and prediction, Monitors.

General Terms

Algorithms, Management, Measurement, Performance, Design, Economics, Reliability, Experimentation, Security, Verification.

Keywords

Cyber-security, testbed, experimental research

1. INTRODUCTION: THE DETER PROJECT

DETER is a research project that is advancing cyber security research practices, by extending the methods, technology, and infrastructure required for scientific development of cyber-defense technology [1], [2]. Our research results are put into practice by operating and evolving DeterLab, transforming it from a basic hardware-focused network security testbed within the early phase of the project, through the shared experiment and testing lab of the middle phase of DeterLab use, towards a new kind of facility for cyber-security experimental science.

Our vision for the next iteration of DeterLab is a facility that is used as a scientific instrument, where researchers create knowledge and understanding through observation, modeling, and measurement of computer and cyber security threats and defenses. We are actively engaged in research and development to improve the scope and usability of this instrument, throughout the life cycle of an experiment – definition, execution, and interpretation. An additional and fundamental goal is that the scientific facility will be to support diverse research communities, and enable users to contribute to and leverage a common body of tools and knowledge. That leverage will enable researchers to build on one another's experimental work, with reproducible experiments and repeatable experimental procedures.

We believe that this evolution of DeterLab will enable shifting the science of cyber security experimentation towards rigorous design, construction, execution, and interpretation of experiments. Such a shift is required to advance the scale, pace, and power of cyber-security research, and to expand the research community and accelerate their work of developing the innovative, scientifically tested, and demonstrably effective new cyber-defenses required to meet challenges ranging from personal digital security to national-scale critical infrastructure protection.

In pursuit of this goal, the DETER project's research program has been influenced by several lessons learned from our experiences in evolving DeterLab, and from supporting and interacting with many researchers using DeterLab. Those lessons have driven several new areas of work on developing cyber-security science methods and technology, that is, the tools, techniques, methodology, resources, and facilities that are needed to provide DeterLab researchers with major advances in their scientific and experimental capabilities: repeatable, flexible, and variable experimentation, discovery, observation, and testing.

This paper provides: some background motivation and history for the DETER project; a review of our key advances in creating a science of experimentation, and lessons learned from these advances; a description of the current DETER research program and roadmap; and a prognosis for the use in DeterLab of the resulting innovations in cyber-security research practices and infrastructure, as we progress toward our vision of a new environment and community for science based cyber security experimentation and test.

2. BACKGROUND: MOTIVATION AND HISTORY

The DETER project's creation grew out of a set of related observations made within the computer and network security research community, funding organizations, and security product companies:

- Security technology and development was largely re-active in nature.
- Security technology development was slower in pace than the evolution of existing threats and the emergence of new threats.
- Successfully and widely deployed security technology (host security, communication security, network boundary control) could be tested with common equipment at small scale.
- Emerging threats, not addressed by deployed security technology, operate at Internet scale (worms, DDOS); requiring radically new classes of defense, and hence radically new evaluation strategies for these defenses, that focus on scale and aggregate behavior.
- New security approaches (e.g., behavioral anomaly analysis) also need large scale and highly varied testing.
- Security innovators lack the facilities to test new security technology in test environments with scale and fidelity to the real deployment environment, and typically construct their own test environment with little or no leverage from the testing work of other innovators.

A consequence of these observations was that promising new security technologies, often from innovators with limited testing resources, fared poorly when tested by applied security practitioners in real deployment environments. [3] In such cases, technology transfer was problematic because of significantly lower effectiveness outside the innovator's limited test facility. In many cases, commercial organizations did not find it cost effective to engage in further development to increase effectiveness.

With this background in 2001-2002, one of the several factors of cyber-security deficiency seemed to be addressable: the lack of testing facilities with significantly greater resources and flexibility than the limited test environments of most innovators, and greater fidelity to real deployment environments. One DARPA-sponsored report [4] called for and stated requirements for a national cyber-defense technology test facility. One result of that report was the impetus for funders at NSF and DHS to define and fund the project that was the first phase of DETER.

The initial focus of DETER was to build such a national testbed, enabling cyber security innovators to test new technology at larger scale, with more complex test fixtures, assembled to be more representative of real deployment environments. The first-phase DETER project (led by USC/ISI, UC Berkeley, and Sparta, Inc.) was funded by two NSF programs – Experimental Infrastructure Network (EIN) and Network Research Testbeds (NRT) – and by DHS. At the same time EIN and NRT co-funded the EMIST project, composed of researchers from Penn State, McAfee Research, ICSI, Purdue, Sparta, Inc., SRI International, and UC Davis. EMIST researchers were to use the DETER testbed, help build knowledge about researcher's needs based on experience working in the testbed, and build experience with existing testing tools used in the testbed. Together these efforts led to the success of the first phase of DETER, with the assembly of the network and physical resources, development of controls and user interfaces for experimenters, assessment and integration of existing tools, and the creation of a collaborative community of researchers.

The testbed became operational in March 2004. The first DETER Community Workshop was held in November 2004, with working groups of researchers who published refereed publications on work performed in the DETER testbed covering, e.g., DDOS defense [5], worm dynamics [6], worm defense [7], and detection of routing infrastructure attacks [8]. The ensuing years saw maturation of the testbed through use and expansion, and growth of the research community with a greatly increased breadth of activity. Both DETER researchers and community collaborators worked on research topics in the technology for supporting and enabling cyber-security research work: experiment automation, benchmarking, scaling via hypervisor usage, malware containment, and our initial work on federation [9], now a central component of DeterLab technology.

In the second phase of DETER, 2007-9, the results of this "research on research" – our exploration of novel technologies and methodologies for cyber-security research – were put into practice in the testbed, which was also expanded in capacity. The result was the evolution from the DETER testbed to DeterLab, a shared virtual lab composed of the underlying testbed resources, technology for using and managing the resources as test fixtures, and a growing variety of tools and services for experiment support, including the full-support release of the federation capability, and the first-generation experimenters' workbench, SEER [10].

With the technological maturity achieved in this phase, and the experience gained from supporting over 1000 researcher team members, the stage was set for DETER project activities to focus increasingly on research and development in the areas of cyber-security experimentation methodology, infrastructure, tools, resource expansion, utilization innovations, and other new methods of using DeterLab for scientific experimentation.

The experience during this phase included several lessons learned that provided important guidance for the current DETER research program. The balance of this paper describes five significant lessons learned, and outlines the current research program developed from the combination of our vision of an advanced scientific instrument, and lessons learned from the developing community.

3. LESSONS LEARNED: CHALLENGES FOR SCIENCE BASED EXPERIMENTATION

Here we describe several lessons learned from early experience with DETER. These lessons derive from observing norms of researcher activity that emerged in the 2nd phase of DeterLab use. Those norms included some basic terms and a typical workflow. Researchers working in DeterLab are called "users" or "experimenters". A team of users working together is called a "project." The focus of activity of a project is a construct called an "experiment" – a term that applies whether the construct is used in the strict sense as part of activity to prove or disprove a hypothesis, or the construct is used for broader purposes such as observing malware behavior, measuring the effect of counter-measures, or other forms of testing or observation that could contribute to hypothesis formation or demonstration of effectiveness of a counter-measure.

The concept of an "experiment" is broad. In DeterLab's vernacular the term "experiment" is used at minimum, to describe the experimental apparatus or environment that users have constructed from computing and network resources, hosts, software, test fixtures, measurement tools, and other fixtures or components of a system that experimenters operate. Beyond this, "experiment" is also frequently used to describe the experimental procedures and experimenter activity to interact with an apparatus in operation, and to review results of operation – that is, the entire "experimental protocol" for a particular research activity.

A large portion of our learning consists of observations and user feedback about the process of designing, building, and carrying out an experiment. Experimenters typically work in a high-level workflow that consists of: initial design and construction of an experimental apparatus; exploratory operation of the apparatus during iterative construction; initial full test runs of operating the apparatus; reviewing and analyzing results of test runs; iterative re-working of the apparatus, towards a fully satisfactory apparatus that repeatably operates as expected, and generates output such as log data, network traces, etc.; analysis of output data to create experimental results. We refer to this workflow as the "experiment lifecycle." Our goal is to change this lifecycle from a manual human intensive *ad hoc* set of processes to a highly automated set of processes tied semantically to an experimenter's model and exploration motivations.

3.1 Lesson Learned: Experiment Construction Considered Difficult

DETER's original tools for construction of experimental apparatus were inherited from Emulab [11], the base technology of the original DETER testbed. These tools provided sophisticated but low-level capabilities for managing the physical computing and network resources of the testbed, and using them to create emulated networks within which an experimenter's activity took place. For our original set of EMIST researchers, this toolset was useful, because they were experienced researchers who valued the "expert mode" in which every detail of a test network could be specified. In addition, many of these early users were familiar in concept or in experience with other network testbeds.

However, we quickly confirmed that the "expert mode only" approach was limiting for many of our researchers, some of whom were less concerned with network-centric security research, and more oriented toward security research that did not depend critically on an exactly specified network environment. Novice DeterLab experimenters with modest research experience faced a steep curve to learn how to create an emulated network of low complexity, but useful for testing. For very experienced cybersecurity researchers starting work in DeterLab, there was also a steep curve to learn how to create an emulated network of moderate complexity and realism sufficient for their work.

One reason for the complexity of network definition lay in the typical first step, preparing a file containing expressions in a specification language to define each individual network node. From "toy network" examples [12] one can extrapolate the level of detail required to specify non-trivial network topologies. In addition to the topology, each node may need to be assigned certain properties so that, e.g., some nodes may serve as background traffic generators or traffic delay nodes. Following topology definition and attribute specification, there are further steps: nodes acting as hosts need to be loaded with a boot image of OS and application software, often followed by the addition of other software, such as experiment-specific software, or packages for monitoring and logging host or network activity. All nodes must be network-configured with network interface devices, IP address, DNS and default route settings, etc.

In addition to the level of effort required, we also had a concern about methodology – in most cases, each experimenter had to do almost all the above activities, with very little leverage of previous experimenters' work, other than perhaps using published network definition files as a model or template. Our own work [13] to build a reference case of a DDOS experiment, designed for re-use and extension by others, was instructive and useful, but also served to highlight the large amount of required detail that was not central to some researchers' work, but was nevertheless pre-requisite to a functioning experiment.

In other words, the experiment definition methodology lacked abstraction and re-use. Acceleration of the pace of cyber-security research was blocked by the necessity of each experimenter needing to specify a great deal of structure, much of which was not critical to their needs, and without recourse to others' work. Our lesson was that the construction part of the experiment lifecycle needed considerable additional automation, new methodology, and supporting features for abstraction, data hiding, and re-use. As our research on higher-level experimental infrastructure support turned to "Experiment Lifecycle Management" (ELM), we added the objective that a new experiment should be able to "stand on the shoulders of previous experiments, rather than standing on their feet".

3.2 Lesson Learned: Diverse and Flexible Federation Considered Valuable

DeterLab's federation capability was originally conceived out of our goal to expand the possible scale of an experiment by enabling an experiment to use not only DeterLab's physical computing and network resources, but also those of other testbed facilities, such as such as Emulab [14], PlanetLab [15], and GENI. [16] The DETER team set up federation arrangements and functionality with such testbeds, both as part of our own research on federation, and also to benefit experimenters seeking larger scale and/or more fidelity by introducing wide-area network topology into an experiment.

However, once the first-generation federation capability was used by DeterLab researchers, we learned of additional benefits sought by them, beyond DETER-managed federation with other testbed facilities. One type of additional benefit was the inclusion in an experiment of unusual or unique resources available in specialized facilities, for example: SCADA systems and embedded controllers, in national labs; supercomputing facilities in high-performance computing facilities; and rare, new, or very large scale networking gear available in labs set up for testing them, such as the University of Wisconsin WAIL [17] facility.

In addition to this "specialized" integration of computing and network resources outside DeterLab, some researchers also sought federate with their own facilities and/or those of their collaborators. Some additional degree of scale-up could be achieved by joining those more ordinary resources "down the hall" from the researcher with the larger scale resources in DeterLab.

These types of desired federation were beyond the scope of the original federation model of linkage between an external network testbed and the testbed underlying DeterLab. To support these types of federation – and others not yet conceived – we began to address issues of resource usage policy and access control, and enriched the federation mechanisms to support them, beyond the originally conceived testbed-to-testbed federation.

3.3 Lesson Learned: Experiment Isolation Considered Limiting

During this same period, we noted that changes in the threat landscape required stark changes to the DETER facility's initial conception of experimental environment. For example, the initial intended methodology for malware experimentation in DETER was to observe and capture malware in the wild, and then to run the captured malware in a simulated network in the testbed, which was fully isolated from the public network by a number of extremely rigid segregation measures [18].

This approach quickly proved limiting for cases where the software-in-the-wild has non-deterministic behavior; because in this case the behavior of a copy in the testbed may have low fidelity to behavior in the wild. Another limitation is a timing issue: for emerging threats, the time required for accurate capture from the wild may introduce delays in the experimenter's ability to test.

As a result, we began to explore a methodology for "controlled Internet access" from DeterLab, in order to explicitly support and control valuable and effective, but also potentially *risky,* experiments – that is, experiments that pose a risk to the outside world, or are at risk from the outside, in addition to the inherent risk of using malware in a testbed or test lab. Some examples of experimentation to be enabled by risky experiment management:

- Place in DeterLab some targets for malware in the wild, and observe in a controlled environment the methods of attack; more expedient than trying to capture the malware and accurately replicate its execution in the test environment. Researchers at CMU and UC Berkeley were some of the first to use the new controlled internet access in order to attract drive-by downloads. The scenario was: a node in DETER visits some Web page, gets infected by malware and that malware instructs it to go visit other Web pages in unpredictable manner. Then they were able to use the infected nodes and behavior to analyze the malware. [19]
- Place in DeterLab some peer computing elements to join in collaborative computing in the wild, for example real anonymity services and infrastructure, the operation of which is dependent on small-time changes in behavior that are non-deterministic; more expedient than replicating a privacy network at scale in a lab, and have the simulated behavior have high fidelity to real-world behavior.
- Place in DeterLab some nodes to serve as bots in botnets, to observe bot/botmaster behavior; more expedient than trying to replicate botmaster behavior with the same software and the same human operator behavior as real botmasters.

The common theme – whether or not malware is used in DeterLab – is that some software of interest has properties that depend on test fixtures or test procedures – specific software, or specific behavior, or human factors – that are difficult to replicate at high fidelity. Partly in response to experimenter requests, and partly from our desire to expand DeterLab's capabilities to accommodate this common theme, we began work on both short-term expedient approaches to controlled internet access and longer-term approaches to flexibly manage this sort of interactions.

The short-term approach involves the use of an *ad hoc*, experiment-specific tunnel node as one node in an experiment apparatus, to permit other nodes to interact with outside systems. The tunnel node is specially prepared by DeterLab network operations, to implement network controls on outside access, but permit the interaction that the experimenter desired and that DeterLab management would permit. Naturally this approach is hardly scalable, requiring the use of scarce operations staff time, and relying on informally stated requirements for outside interaction.

Recognizing the limits of this approach, we also began work on a methodology for more flexibly managing CIA, by transforming a risky experiment into a safe experiment. Our work builds on a single, simple fundamental observation:

- If the behavior of an experiment is completely un constrained, the behavior of the host testbed must be completely constraining, because it can assume nothing about the experiment.
- However, if the behavior of the experiment is constrained in some known and well-chosen way or ways, the behavior of the testbed can be less constraining, because the combination of experiment and testbed constraints together can provide the required overall assurance of good behavior.

We refer to this approach as Risky Experiment Management (REM) T1-T2 because it combines two sets of constraints, derived from the above observation, to limit the overall risk of the experiment. We call the first sort of constraints "experiment constraints" or "T1 constraints"; these are constraints naturally exhibited or explicitly imposed on the experiment. We call the second class of constraints "testbed constraints" or "T2 constraints"; these are constraints imposed by the testbed itself. We often refer to overall concept as the "T1/T2 model."

Implementation of the REM-T1/T2 approach [20] will require tools for formal definition of the experimenter's requirements – defining the T1 transformation – and methods and automation for defining the additional constraints that define the T2 transformation. These advances will be required for risky experiments to be defined, controlled, and permitted with more assurance than experiment-specific tunnel nodes. Section 4.4 provides further information on future efforts on REM-T1-T2.

3.4 Lesson Learned: A Requirement for Scale and Fidelity Flexibility

In the initial DETER testbed, the basic approach to scalability rested on scaling up the amount of available hardware, and on a fixed set of approaches to fidelity using individual nodes. By "fidelity" we mean that in a testbed there is a set of nodes that exist to model the behavior of nodes in the real world, and the testbed nodes' behavior in the simulated environment is behavior that is similar to real nodes in real environments. The fixed set of approaches to fidelity, in this classic approach, is a range from a single testbed node acting like a single real node, with high fidelity; to a single testbed node standing for a large number of internal network end-nodes.

In this approach, the maximum size of an experiment is essentially bounded by the number of hardware nodes. As we built out DeterLab, not only did we want to increase the scaling of the hardware, but we also recognized that many of our users' experiments did not require high fidelity in every part of the experimental apparatus. We recognized a class of "multi-resolution" experiments [21] in which:

- some parts of an apparatus require high-resolution nodes with high fidelity;
- some other parts require a lower degree of resolution and can represent real computing at a larger scale;
- there is a "scale of scaling" with points that range from high fidelity and linear scaling, to low fidelity and high scalability;
- different points on the scale will be enabled by different mechanisms for emulation and simulation.

As a result of this observation, we began to explore methods to incorporate a number of representation methods that together provide a full *spectrum* of scale-fidelity tradeoffs for experimental system components. The following is a partial list of examples:

- a single hardware node running a single experiment node, either natively, or via a conventional Virtual Machine Manager (VMM) supporting a single guest OS;
- a single hardware node running several virtualized experiment nodes, each a full-blown conventional Virtual Machine (VM) on a conventional VMM;
- a single node running a large number of lightweight VMs on a VMM designed for scaling the number of experiment-nodes with limited functionality;
- representation of individual experiment nodes as threads of execution in a large-scale thread management environment;
- large-scale software-based network simulation [22].

Further, we recognized that these methods would be more useful to experimenters if all methods were part of a unified framework for the construction of composable experiment apparatus, using both some common building blocks and methods of composition with abstraction and re-use. Our approach to such a framework is to base on it on an abstract fundamental building block called a "container" which represents experimental elements at the same level of abstraction, and is the basic unit of composition for constructing an experimental apparatus. The container-based methodology is a key part of pursuing some important goals:

- leverage DeterLab's physical resources more flexibly to create larger scale experiments;
- enable experimenters to model complex systems with high resolution and fidelity for the things that matter most to them, and abstract out the less important elements;
- reduce the experimenter's workload of experiment apparatus construction, enabling larger scale apparatus with lower levels of effort.

3.5 Lesson Learned: From Experimental Apparatus to Experimental Data to Experimental Results

The previous four lessons learned were largely related to the static aspects of setting up an experimental apparatus: basic construction of an apparatus; use of fixtures for federation; use of fixtures to enable limited communication outside the testbed; and use of fixtures that support orders-of-magnitude experiment scale-up over that obtainable with more simplistic use of physical resources.

Other lessons learned were about the dynamic aspect of running an experiment. Early in the 2nd phase, we recognized the need for a workbench that experimenters could use to operate an experimental apparatus, feeding it input data and events, observing its operation, and adjusting the fixtures for collecting experimental data. The first-phase workbench, SEER [10], met that need to some extent. However, adoption of SEER also brought into focus a growing need for DeterLab experimenters: an approach to the "big data" problem. As DeterLab facilities have matured with scale and power and data capture capability, and as observation of the behavior of a running experiment drove improvements in data collection, the result was, for many experiments, a much larger set of output data to be analyzed from each experiment run.

Further, not only the size of data grew, but also the structure and complexity of the datasets increased. In addition to log analysis tools to help deal with raw data size, there was a need for other methods – and automated support for them – to analyze data in terms of the intended semantics of the experiment run, and ultimately to proceed from data analysis to actual experimental results: proving or disproving a hypothesis, or stating knowledge of malware behavior, or use of metrics for effectiveness of countermeasures.

In other words, experimenters need both tools and methodologies for mining experimental data to discover experiment results. This lesson learned served to underscore the importance of our research work on narrowing this large "semantic gap" as part of our research efforts on Experiment Lifecycle Management.

4. CURRENT DETER RESEARCH PROGRAM

Our current research program includes, but is not limited to, activities related to the above lessons learned. Current research is in some cases an outgrowth of work performed as part of our agenda to enrich the testbed with progressive enhancements that resulted in what we now call DeterLab. During the 2nd phase in which we were learning from DeterLab users, our own enhancement efforts included:

- First generation of federation capabilities [9];
- Risky experiment management and abilities to include outside communication [16];
- The first-generation "experimenter workbench" for managing an experiment in process, viewing its activity and results data [10].

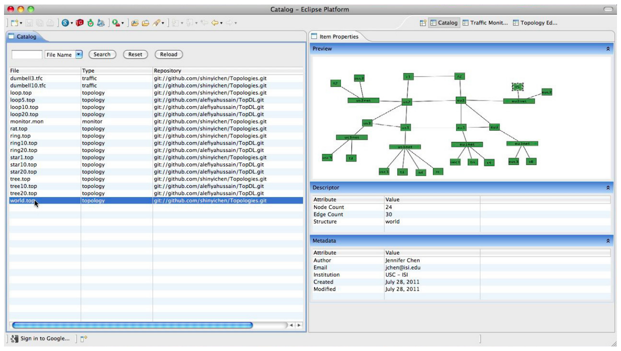
In some cases, there was real synchronicity between our objectives and the needs of DeterLab experimenters. As described above, the first generation of federation capability arose from our desire to reach greater scale by using resources in other testbeds that we could link to; in addition, we learned that experimenters wished to link into their experiments some outside resources of their own, and/or specialized resources that they had access to. As a result, our research agenda (for federation with access control) was enriched with new use cases and additional requirements.

4.1 Experiment Lifecycle Management

Experiment lifecycle management is an outgrowth of work on our first generation workbench, SEER. Indeed, many of SEER's capabilities, including experiment monitoring and visualization, are carried over into the next generation workbench, the

Experiment Lifecycle Manager (ELM), albeit in a new usage paradigm.

One critical aspect of ELM focuses on the general concept of objects that an experimenter uses. DeterLab has grown to include a large number and variety of objects available to experiments. With that growth has come the challenges of giving experimenters the tools need to effectively manage their working set, and (critically) to effectively share with other experimenters. The objects used by an experimenter include scientific, physical, communication, and computational resources used in an experiment. Also included are models, designs, procedures, programs, and data. Storage, presentation, archival, browsing, and searching are basic ELM functions for managing an experiment's components – and allowing other researchers to access them – far beyond the original testbed approach of shell login and filesystem access. We are building this basic management framework on the Eclipse [23] platform, in order to leverage and build upon the many integrated development environment (IDE) capabilities of Eclipse.



File	Type	Repository
dumbell3.tfc	traffic	git://github.com/shinyichen/Topologies.git
dumbell10.tfc	traffic	git://github.com/shinyichen/Topologies.git
loop.top	topology	git://github.com/shinyichen/Topologies.git
loop5.top	topology	git://github.com/alefiyahussain/TopDL.git
loop10.top	topology	git://github.com/alefiyahussain/TopDL.git
loop20.top	topology	git://github.com/alefiyahussain/TopDL.git
monitor.mon	monitor	git://github.com/shinyichen/Topologies.git
rat.top	topology	git://github.com/shinyichen/Topologies.git
ring.top	topology	git://github.com/shinyichen/Topologies.git
ring10.top	topology	git://github.com/alefiyahussain/TopDL.git
ring20.top	topology	git://github.com/alefiyahussain/TopDL.git
star1.top	topology	git://github.com/shinyichen/Topologies.git
star10.top	topology	git://github.com/alefiyahussain/TopDL.git
star20.top	topology	git://github.com/alefiyahussain/TopDL.git
tree.top	topology	git://github.com/shinyichen/Topologies.git
tree10.top	topology	git://github.com/alefiyahussain/TopDL.git
tree20.top	topology	git://github.com/alefiyahussain/TopDL.git
world.top	topology	git://github.com/shinyichen/Topologies.git

Descriptor

Attribute	Value
Node Count	24
Edge Count	30
Structure	world

Metadata

Attribute	Value
Author	Jennifer Chen
Email	jchen@isi.edu
Institution	USC – ISI
Created	July 28, 2011
Modified	July 28, 2011

Figure 1: Screenshot of an experimenter using ELM to view a catalog of experiment components, and select and view a network topology displayed visually

New levels of abstraction in experiment definition are also a key component of ELM. In the original testbed approach, experimenters had to specify in detail a number of different types of resources:

- Computational elements such as physical or virtual hosts, and the complete "network plumbing" configuration of each.
- Elements of a network operating environment, including network topology, router and switch nodes and their configurations.
- Network nodes that perform traffic shaping to simulate real world network conditions, delays, throughput limits, etc.

In addition, experimenters had to specify in detail a number experiment elements running within the network and computational resources: host operating systems, guest operating systems for VMs, application software, and logging and other infrastructure software typical of real systems. Further, experimenters had to deploy on these systems a number of experimental fixtures such as traffic generators, tools for running experimental procedures and collecting result data, and often malware to be observed and cyber-defenses to be tested.

Perhaps most significantly, each experimenter tended to do their own apparatus construction largely from the ground up, with limited leverage of others' work in defining experimental apparatus. In ELM, all these types of experiment resources, elements, fixtures, and artifacts do of course need to be managed as individual objects, as building blocks for components of an experiment. More importantly, we're working on construction methods that include both the basic building blocks, and also structures of them that prior experimenters have contributed. In other words, with ELM, experiments can be highly modular, and explicitly structured for re-use as shown in Figure 1.

Although the detail-oriented "expert mode" is still available, we expect most researchers to use the newer facilities for defining an experiment's components abstractly, with requirements, constraints, and invariants, rather than specify directly and in every detail. For example, an earlier experiment may already have defined an apparatus that simulates a handful of large enterprise networks connected over the public network, a number of ISP networks, and home computers. This apparatus, though conceived for use in worm spread, may nevertheless be described with meta-data that enables a later researcher to identify it as a suitable starting point for their work. The later researcher should be able to use the archived design, and state some new requirements and constraints relevant to their work, or specify some properties of specific experiment fixtures for input generation or monitoring. Without having to know other detail beyond their requirements, experimenters can describe an experiment apparatus entirely independent of its realization on computing and network resources.

Thus far, the description of ELM is analogous to an IDE with source code repositories, modules, libraries, facilities for combining them, with shared storage, versioning, and change control – all valuable advances from the early DETER testbed. However, ELM also provides other critical facilities analogous to an IDE:

- Mechanisms for "realizing" an abstract, modular experiment definition by allocating and configuring real network and computing elements.
- Tools for interpreting experimental data to yield information that expresses experimental results in terms of the experiment's model and the abstractions that helped define the apparatus.

Following sections describe some of our work on advances in realizing and running experiments at scale, and on model-based experimentation that enables semantic analysis of results. That work is directly reflected into the ELM methodologies and tools mentioned above.

4.2 Containers: Scale-up and Flexible Fidelity

Our continuing work on scalability is based on the observations (summarized in Section 3.4) about trade-offs between the fidelity or realism of a computational element in DeterLab, and the scale of network and computing resources required to realize a computational element. However, re-usability is also an important goal for the ease of use of DeterLab tools for constructing an experimental apparatus. By adding new types of computational element (conventional VMs, QEMU lightweight VMs, processes on conventional OSs, QEMU processes, individual threads of execution), each of which can be used to model a node in a simulated network, we added both flexibility and complexity to the methods of constructing an apparatus.

To manage complexity and increase ease of construction, we are developing an apparatus framework centered on an abstraction that we call a "container" [21]. In our new construction methodology, a container is the fundamental building block. A single container may support one or multiple components (elements) of an experimental apparatus, and implements an abstraction layer that hides the details of the inner components, when that container is itself placed inside another container. Figure 2 shows a simple container that contains no other containers, containing only 2 concrete computing elements, such as a VM or thread. Abstraction is provided by the container's communication mechanism, which both connects the contained elements with one another, and also presents an entry/exit point for communication into the container; the communication mechanism advertises to other containers the properties of its container.

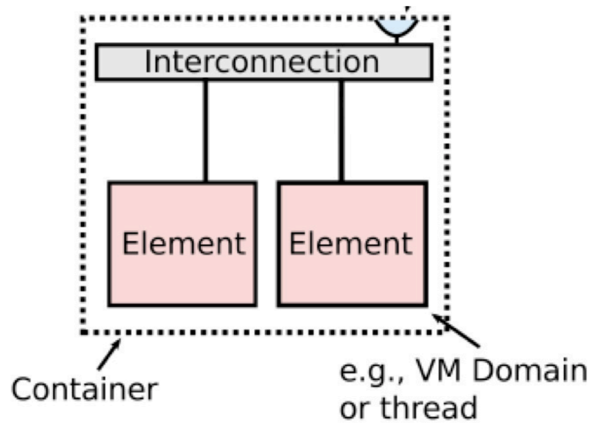


Figure 2: A simple container of two basic computing resources

Abstraction is a central point for continuing to expand the scalability options in DeterLab. Other researchers frequently create new techniques for scalable computing, or new methods of using virtualization or simulation, or performing a form of lightweight computation. Our goal going forward is to incorporate promising work in this area, defining a new abstract type of basic computing element, creating a standard interface for a containerized component based on each new technology, and expressing its tradeoffs explicitly, for use in construction tools.

Thus far, our containers work has been successful for scalability for multi-resolution experiments, and has illustrated the requirements for integrating container technology into the ELM workbench. In addition to the expansion of available basic computing elements described above, there are several areas of ongoing work.

Apparatus construction: We demonstrated feasible levels of scalability and complexity by creating several experiment apparatus, the largest containing over 600 components that were then realized on 8 physical computers and is capable of representing 50,000 computers at a coarse granularity in a specific scenario. The low end of this spectrum would be a modest sized mixed-resolution experiment. The high end would be a large experiment in which all but a few elements had low fidelity. However, construction involved manually matching each element in a desired network topology with a specific container. Clearly, there is promising work to do in automating this process, and integrating containers into the apparatus construction mechanism of our new workbench, ELM.

Re-usability and embedding: ELM provides the ability of experimenters to archive experimental apparatus definitions, or components of them, and for other experimenters to use these archived items as building block for the construction of a new apparatus definition. The definitions can then be used with core DeterLab embedder capabilities for realizing the definition in a real apparatus composed of DeterLab network and computing resources. Again, there is work to do building the workbench technology for containers being one of the objects that can be archived and reused. Likewise, there is work to do with the embedder, to automate realization with little input from experimenters, while also giving experimenters visibility on embedding so that they can vary some of resource utilization or vary the fidelity/scale tradeoff points for a specific apparatus.

4.3 Model Based Experimentation

As described in Section 3.5, DeterLab experimenters have a “big data” problem that will only grow as DeterLab magnifies the scale available to experimenters, and the breadth of tools for collecting experimental data. Our approach to this problem has been to completely re-conceive the methodology for how cyber-security experiments are defined in order to yield data to be analyzed for information that comprises experimental results.

The basis for this approach is no more or less than adopting basic ideas from other experimental sciences that are more mature than experimental cyber-security is at present. The conceptual starting point of an experiment is a real-world situation that displays an interesting problem that is inconvenient to investigate *in situ* in the real world. Instead, we define a conceptual model of the situation, and begin to define laboratory activity that allows us to construct in the lab a physical (or chemical, or biological, or informatic) model of the real-world situation. Part of the function of this model is to serve as a design for an experimental apparatus that the experimenter will observe or modify in order to make inferences from lab observations to the real world where analogous modifications may create analogous results.

This common methodological framework is, however, somewhat unusual for some areas of computer science, where much research is in fact *in situ* – modify an actual computational or communication system to observe whether it better meets the researcher’s goal for speed, efficiency, data access rates, power utilization, etc. Cyber-security research, however, is very definitely model-based. The real world has a host of large-scale systems with complex structures with vulnerabilities and potential mitigations. Where experimental modification of real systems (for example, induced cyber attack) is not feasible, we use a lab environment to create a model, an apparatus to approximate the model, experimental procedures to observe or induce changes in the apparatus, and so on.

However, the early stage use of the DETER testbed was not significantly model-based. Figure 3 is somewhat whimsical but accurate view of experimentation in which modeling is entirely mental, with no externally visible relation between the model and an *ad hoc* constructed apparatus, or its operation. The lab procedures are in fact quite valuable to the researcher, but *ad hoc*, and difficult to document or to be repeated by others. Nowhere is the *ad hoc* nature more evident in the research’s unique ability to pore over network traces to find variations in worm behavior that may be attributed to worm propagation countermeasures.

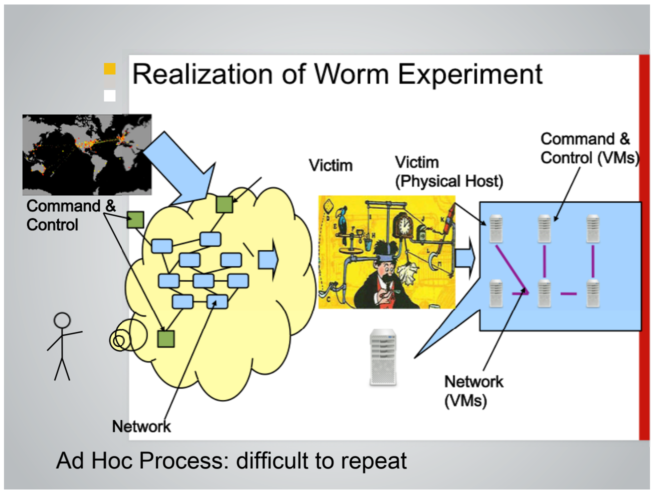


Figure 3: An informal experiment model leading to an *ad-hoc* experiment apparatus

The DETER research program includes work to assist researchers in defining several model-related structures that become part of the methodology for building experimental apparatus, defining experimental procedures, and analyzing experimental data for expected (or unexpected) patterns or changes predicted by the model or a hypothesis derived from it. One purpose of modeling and semantic definition techniques is to *define specific measurements and expectations* to be sought for in the results of an experiment's operation in the lab.

This model-based approach requires new cyber-security methodology and new lab technology, integrated into the already-described experiment lifecycle facilities, but oriented to defining semantics for an experiment and its results, validating an experimental apparatus, and extracting understanding from results. Several types of tools under investigation can potential benefit:

- semantic mechanisms to capture the intent of the experimenter;
- support for monitoring this intent and distributed execution with breakpoints;

abstraction and modeling techniques for experiment design, realization, visualization, and analysis.

The use of these tools is inherently iterative, shown in Figure 4. An experimenter **defines** a model, an apparatus to implement it, procedures to operate it; then runs the experiment by operating the apparatus, **executing** software or manual steps to perform experimental procedures; the resulting data is **interpreted** to extract information, which can then be used to iterate on the experimental apparatus, measurement scheme, or procedures.

To date, most experimentation presumed the existence of some form of model of the system under test that the experimenter uses to map his experiment objectives onto an apparatus in an experimental facility such as DeterLab. While this has often been true for the low-abstraction-level tasks of defining network topologies and traffic generator configurations, the lack of rigor in the initial steps often undercut the efficacy of the entire experimental process by providing little guidance or expectation for the resulting experimental data, and no ability for knowledge-based iteration.

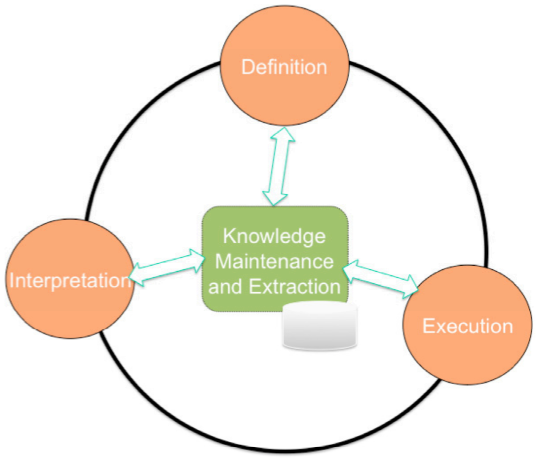
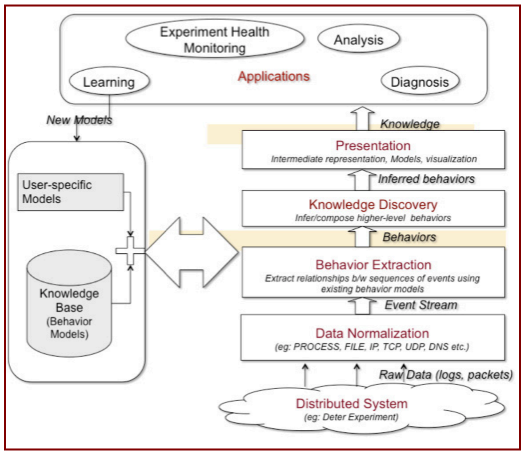


Figure 4: The iterative aspect of the experiment lifecycle

As in any scientific discipline, often the greatest challenge lies in creating an appropriate representation of the object for study, representative across the measurement dimensions that matter, while carefully documenting the simplifying assumptions and abstractions that are made in the process. While the most general case of this problem is very hard, we are working to extend DeterLab's experimenter support back into the early reasoning process of experiment design. We approach this through a set of Model Based Scenario development techniques, in which experiments are constructed from a basis in general models that capture the behavior of different dimensions of cyber security experiments.



Experiment Health Monitoring
Analysis
Learning
Applications
Diagnosis
New Models
Knowledge
Presentation
Intermediate representation, Models, visualization
Inferred behaviors
User-specific Models
Knowledge Discovery
Infer/compose higher-level behaviors
Behaviors
Knowledge Base (Behavior Models)
Behavior Extraction
Extract relationships b/w sequences of events using existing behavior models
Event Stream
Data Normalization
(eg: PROCESS, FILE, IP, TCP, UDP, DNS etc.)
Raw Data (logs, packets)
Distributed System
(eg: Deter Experiment)

Figure 5: Development of experimental knowledge from experiment data

Using the workbench and tools that we are investigating, an experimenter may be able to refine the models into apparatus templates or experiment-procedure definition or recipes, which can be used to plan data analysis. The analysis would not be bottom up or *ad hoc* pattern based, but rather following a *knowledge discovery* procedure (shown in the middle of Figure 5) that is derived from the model and its various components and formalized assumptions such as expected behavioral invariants or functional constraints. In other words, we are working towards a shift in methodology where new tools assist experimenters in rigorous construction, execution, and interpretation of *semantically validated* experiment design and execution.

During the specification phase of an experiment, invariants associated with the model will be used to verify that the experiment being developed is internally consistent. During execution, invariants will be used to ensure that the intended experimental semantics are realized (*validity management).* Finally, invariants will be invoked as part of the interpretation and visualization of results – providing methods for the experimenter to tackle large amounts of data in order to determine whether an experiment run produced expected results or potentially interesting unexpected results.

A simple example is the development of a model state space for execution of (and potential attack on) a communication protocol. A variety of data (packet dumps, web server logs, auth logs) can be normalized for input into analysis and visualization tools that assist the experimenter in mapping from actual events to expected behaviors. Figure 6 shows a conceptual view of the model state space, with various possible paths through it; a path to the "success" node would be expected results of experiment execution (visible in detail in event logs), while other paths indicate a violation of an assumption about correct behavior, which may be detectable sign of an attack or malfunction (accompanied by a particular reason for the violation, attributable to event logs).

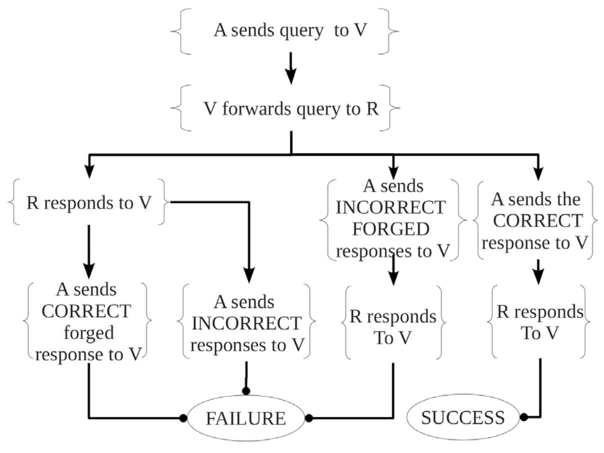

A sends query to V
V forwards query to R
R responds to V
A sends INCORRECT FORGED responses to V
A sends the CORRECT response to V
A sends CORRECT forged response to V
A sends INCORRECT responses to V
R responds To V
R responds To V
FAILURE
SUCCESS

Figure 6: An example of a semantic model for an experiment

Model based experimentation takes on an increasing importance when designing experiments that span both cyber and physical elements. The physical components are likely based in some set of models (real world, empirical, or theoretical). In order to capture the interactions and relations between the cyber and physical, it will be necessary to compose models. Recent work in Secure Smart Grid Architectures [24] argues that:

> "An analysis of the cyber-physical security of a smart grid architecture must focus on the impact of faults and interactions that cross domains rather than the localized response that might be seen in traditional penetration testing. This requires a capability to model large scale response to cyber-attack, as well as to perform modeling or simulation of the physical components of a system."

The Smart Grid security team at USC-ISI and JPL are currently using DeterLab to develop a secure smart grid architecture. They ave created a taxonomy of cyber and physical threats and are exploring federation of DeterLab with other labs and testbeds that provide physical simulation and emulation tools for modeling the systemic response of the grid. Such experiments will span multiple sites and will enable the use of specialized resources to participate in large-scale experiments.

We view current research efforts such as the Smart Grid and other emerging cyber physical domains as new use cases for examining and validating the evolving features and capabilities of the DeterLab that we are developing as part of the DETER project research program.

4.4 Additional Directions

The three previous sections have outlined some of the key areas of our current research work, some of which was guided by lessons learned, in addition to the results of our own research. The research program also includes areas for future work, in which we will extend the results of earlier topics by applying to them some of the advances that we are making in our current research work.

Risky experiment management is one area of prior work that also occupies a place in the roadmap. To put into practice the management approach described in Section 3.3, we will need to (a) develop DeterLab facilities for an experimenter to develop and refine specifications of their experiment's requirements for Controlled Internet Access, and (b) develop automation tools to create an experiment-specific gateway node. The automation tools will need to both implement the experimenter's requirements, and also implement DeterLab's constraints defined in the T1/T2 approached described in Section 3.3.

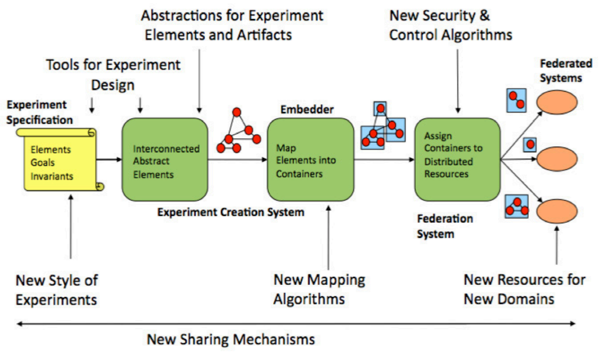
For risky experiment management, this future elaboration will depend on the results of two areas of current research activity. The modeling and specification work (described in Section 4.3) will provide key elements of the experimenter facility to define constraints and invariants on the experiment's communication via controlled internet access. The container work (described in Section 4.2) will enable DETER project research staff to create reusable building blocks for gateway implementation, each with advertisements that will assist the automation tools in constructing a container to serve as a gateway node that implements the required controls for controlled internet access as needed by the particular experiment.

A second part of the research roadmap is elaboration of prior work on federation, to support a new form of DeterLab experimentation. A multi-party experiment is one in which the experimental apparatus is built from sub-components that are federated to create the whole, and each federant has complete information only about their own sub-component, with only partial information about other sub-components. This form of experiment can be used to model several different kinds of cyber-defense situations: adversarial situations (e.g. red-team/blue-team exercises); realistic forensic or defense scenarios (e.g., attack target with limited information about attacker); or partial collaboration situations in which separate organizations collaborate on defense without granting full visibility to collaborators.

Support for multi-party experimentation will depend on the current full-production federation capability in DeterLab, and the results of several areas of current DETER research: modeling and specification work (described in Section 4.3) to state constraints and invariants on activities of each party; and the container work (described in Section 4.2), which is essential to scale out each party's sub-apparatus to realistic proportions needed for the types of multi-party experiments currently envisioned.

4.5 Integrating the Pieces: Towards a New Experimental Cybersecurity Research Paradigm

The above areas of current research and future research roadmap provide the foundation for our program towards new science based experimental cybersecurity. Our focus is extending DeterLab with new capabilities resulting from work in these areas, as well as integrating the new and existing capabilities. The integration is critical, including functional integration with the new ELM workbench; but more important is integration into a new *methodology* for the experiment lifecycle. Five of several possible lifecycle phases are illustrated in Figure 7:



Abstractions for Experiment Elements and Artifacts
New Security & Control Algorithms
Tools for Experiment Design
Federated Systems
Experiment Specification
Embedder
Elements
Goals
Invariants
Interconnected Abstract Elements
Map Elements into Containers
Assign Containers to Distributed Resources
Experiment Creation System
Federation System
New Style of Experiments
New Mapping Algorithms
New Resources for New Domains
New Sharing Mechanisms

Figure 7: New cyber-security research methodologies

These are

- a new methodology for specifying experiments, including model-based specification, and elements of previous experiment descriptions;
- new tools to completely flesh out the structure of an experiment, with only the essential elements and abstractions;
- new technology for realizing the conceptual structure of an experiment, by embedding it in a subset of DeterLab's real and virtual resources for computation and networking;
- new facilities and new controls that enable larger scale and more flexible use of federated systems and domain-specific resources – especially domain-specific resources that are available via federation; and
- across all of these areas, new mechanisms and facilities to share experiment building blocks among experimenters, who can accelerate their experiment-creation work using the results and knowledge gained by previous work in DeterLab.

As we gain experience with this integration, we expect that we and DeterLab experimenters will develop cyber-security experimentation methodologies that can help to accelerate the pace of cyber-security innovation, and also dramatically improve the scientifically demonstrated effectiveness of innovations as they move from the lab into practical use.

5. SUMMARY

The DETER project is continuing with its research program, much of which has been described in brief in this paper. Our research target continues to be the technology and methodology needed for the practice of cyber-security research to become an experimental science, with rigorous experiment design, construction, execution, interpretation, sharing, and repeatability of experiments. We believe that such a transformation of cyber-security research is needed to expand and accelerate the research community's efforts.

Expansion and acceleration are becoming increasingly important, as inter-connected cyber systems continue to expand into nearly every area of human infrastructure, from critical infrastructure for communication, finance, transportation, and power, to networked computing systems that are intimately integrated into our lives: our vehicles, our handheld wireless personal computing and communications ("smart phones"), regulation of our homes' utility use ("smart meters"), remote control and regulation of our homes' fixtures ("smart grid") including safety-critical appliances, as well as our healthcare, and even our bodies, with networked controllers for implanted medical devices.

From national and international security to personal safety, the number and variety of targets continues to increase along with an accelerating expansion of the total attack surface available to adversaries who have an increasingly powerful portfolio of tools and customers. When the DETER project started, cyber-security technology development and technology transfer were often resource intensive, and often lacking in proactive approaches for asset protection to be sufficient to increase the level of cyber defense of critical assets. In the past 8 years, we have built some significant infrastructure for the development and test of new cyber-defenses. As we look ahead to the coming years, we expect that transformation of research tools and methods will contribute to the much needed expansion and acceleration of research, which can lead to an accelerated pace of deployment of scientifically tested and effective cyber-defenses.

Continuing maturation of the DeterLab facility is necessary, but so is the accelerated growth of a cyber-security research and test community that can rapidly leverage one another's work. In addition to the research and methods described in this paper, further development of the cyber-security experiment science community is a critical shared responsibility of the larger cyber-security community.

6. ACKNOWLEDGEMENTS

This paper builds on efforts at ISI over the past 8 years and would not have been possible without the contributions of the entire DETER team. Special recognition is given to: John Wroclawski for setting an ambitious research program; Ted Faber, Mike Ryan, Alefiya Hussain and Jelena Mirkovic for delivering new capabilities under that program; to Bob Braden for careful crafting of proposals, capturing all of the prior work and proposed new work; and to John Sebes amanuensis extraordinaire.

This research was sponsored by the US Department of Homeland Security and the Space and Naval Warfare Systems Center, San Diego (contract number N66001-10-C-2018), and the US National Science Foundation (contract number CNS-0831491). Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the Department of Homeland Security or the Space and Naval Warfare Systems Center, San Diego.

7. REFERENCES

[1] Current Developments in DETER Cybersecurity Testbed Technology. T. Benzel, B. Braden, T. Faber, J. Mirkovic, S. Schwab, K. Sollins and J. Wroclawski. In *Proceedings of the Cybersecurity Applications & Technology Conference For Homeland Security (CATCH 2009)*, March 2009.

[2] The DETER Project - Advancing the Science of Cyber Security Experimentation and Test. Terry Benzel, Jelena Mirkovic, et al. *IEEE HST 2010 Conf*, Boston, MA, November 2010.

[3] Vulnerability Detection Systems: Think Cyborg, Not Robot. S. Heelan. In *IEEE Security and Privacy*, special issue "The Science of Security," Vol. 9, Issue 3, May/June 2011.

[4] Justification and Requirements for a National DDoS Defense Technology Evaluation Facility. W. Hardaker, D. Kindred, R. Ostrenga, D. Sterne, R. Thomas, Network Associates Laboratories Report, July 26, 2002.

[5] Cyber defense technology networking and evaluation. R. Bajcsy, T. Benzel, M. Bishop, B. Braden, C. Brodley, S. Fahmy, S. Floyd, W. Hardaker, A. Joseph, G. Kesidis, K. Levitt, B. Lindell, P. Liu, D. Miller, R. Mundy, C. Neuman, R. Ostrenga, V. Paxson, P. Porras, C. Rosenberg, J. D. Tygar, S. Sastry, D. Sterne, S. F. Wu. In *Communications of the ACM*, Special issue on "Emerging Technologies for Homeland Security," Vol. 47, Issue 3, pp 58-61, March 2004.

[6] Preliminary results using scale-down to explore worm dynamics. Nicholas Weaver, Ihab Hamadeh, George Kesidis and Vern Paxson. In *Proceedings of the 2004 ACM workshop on Rapid Malcode*, pp. 65-72, 2004.

[7] A hybrid quarantine defense. P. Porras, L. Biesemeister, K. Levitt, J. Rowe, K. Skinner, A. Ting, In *Proceedings of ACM WORM*, Washington, DC, Oct. 29, 2004.

[8] Combining visual and automated data mining for near-real-time anomaly detection and analysis in BGP, S.T. Teoh, K. Zhang, S.-M. Tseng, K.-L. Ma and S. F. Wu, In *Proceedings of ACM VizSEC/CMSEC-04*, Washington, DC, Oct. 29, 2004.

[9] A Federated Experiment Environment for Emulab-based Testbeds. T. Faber and J. Wroclawski, in *Proceedings of Tridentcom*, 2009.

[10] SEER: A Security Experimentation EnviRonment for DETER. Stephen Schwab, Brett Wilson, Calvin Ko, and Alefiya Hussain, in *Proceedings of the DETER Community Workshop on Cyber Security Experimentation and Test*, August 2007.

[11] Emulab Testbed Web site, http://www.emulab.net

[12] DeterLab Testbed wiki, https://trac.deterlab.net/wiki/Topologies

[13] DDoS Experiment Methodology. Alefiya Hussain, Stephen Schwab, Roshan Thomas, Sonia Fahmy and Jelena Mirkovic. In *Proceedings of the DETER Community Workshop on Cyber Security Experimentation and Test*, June 2006.

[14] *An Integrated Experimental Environment for Distributed Systems and Networks*. White, Lepreau, Stoller, Ricci, Guruprasad, Newbold, Hibler, Barb, and Joglekar, in *Proceedings of OSDI 2002*, December 2002.

[15] Operating System Support for Planetary-Scale Network Services. A. Bavier, M. Bowman, B. Chun, D. Culler, S. Karlin, S. Muir, L. Peterson, T. Roscoe, T. Spalink, and M. Wawrzoniak. (*NSDI '04),* May 200

[16] Larry Peterson, John Wroclawski (Eds), "Overview of the GENI Architecture,"GENI Design Document 06-11, Facility Architecture Working Group, September 2006

[17] Bench-style Network Research in an Internet Instance Laboratory. Paul Barford, Larry Landweber, in *Proceedings of SPIE ITCom*, Boston, MA, August, 2002.

[18] A Plan for Malware Containment in the DETER Testbed. Ron Ostrenga and Stephen Schwab, Robert Braden. In *Proceedings of the DETER Community Workshop on Cyber Security Experimentation and Test*, August 2007

[19] BitBlaze: A New Approach to Computer Security via Binary Analysis. Dawn Song, David Brumley, Heng Yin, Juan Caballero, Ivan Jager, Min Gyung Kang, Zhenkai Liang, James Newsome, Pongsin Poosankam, and Prateek Saxena. In *Proceedings of the 4th International Conference on Information Systems Security, Keynote Invited Paper.* December 2008.

[20] A Two-Constraint Approach to Risky Cybersecurity Experiment Management. J. Wroclawski, J. Mirkovic, T. Faber and S. Schwab, Invited paper at the Sarnoff Symposium, April 2008.

[21] Building Apparatus for Multi-resolution Networking Experiments Using Containers. Ted Faber, Mike Ryan, and John Wroclawski, in submission.

[22] Scalability and Accuracy in a Large-Scale Network Emulator. Amin Vahdat, Ken Yocum, Kevin Walsh, Priya Mahadevan, Dejan Kostic, Jeff Chase, and David Becker. *Proceedings of 5th Symposium on Operating Systems Design and Implementation (OSDI)*, December 2002.

[23] *Practical Eclipse Rich Client Platform Projects* (1st ed.). Vladimir Silva (11 March 2009). Apress. p. 352. ISBN 1430218274.

[24] Mediating Cyber and Physical Threat Propagation in Secure Smart Grid Architectures. Clifford Neuman and Kymie Tan, to appear in *Proceedings of the 2nd International Conference on Smart Grid Communications (IEEE SmartGridComm),* October 2011, Brussels.

Facing the Facts about Image Type in Recognition-Based Graphical Passwords

Max Hlywa
Department of Psychology
Carleton University
Ottawa, Canada
mhlywa@connect.carleton.ca

Robert Biddle
School of Computer Science
Carleton University
Ottawa, Canada
robert_biddle@carleton.ca

Andrew S. Patrick
Department of Psychology
Carleton University
Ottawa, Canada
andrew@andrewpatrick.ca

ABSTRACT
Graphical passwords are a novel method of knowledge-based authentication that shows promise for improved usability and memorability. This paper presents two studies that examined the effect of image type in cognometric, recognition-based graphical passwords. Specifically, the usability of such authentication schemes was explored at security levels equivalent to those acceptable for text passwords. Related psychological theory was drawn upon to consider the relative strength of visual memory, to distinguish recognition from recall, and for face recognition by humans. With image type as the independent variable, login success and login time were observed as the dependent variables. Results from both studies showed that participants in the object images condition performed equal to or better than those in the face images condition. Importantly, there was no evidence to support the claim that the use of face images in the authentication scheme would result in superior user performance.

Categories and Subject Descriptors
K.6.5 [Management of computing and information systems]: Security and protection: Authentication

General Terms
Security, Human Factors, Visual memory

Keywords
Authentication, graphical passwords, usable security

1. INTRODUCTION

The effectiveness of any computer security system depends on proper use. Security experts will sometimes refer to people as "the weakest link in the chain" of system security [32]. While it is true that security systems are often rendered ineffective because users fail to use them properly, this failure can also be seen from the perspective that weak system design is to blame. One of the reasons that current security systems suffer is because they often fail to incorporate human factors knowledge in their design. As humans, we have cognitive limitations that restrict and define the potential for our interaction with computers. System designs that fail to take these human factors into account will inevitably lead to failure. However, potentially more constructive than a review of the limitations of our cognitive capacity is to consider and leverage its strengths.

Today, the most commonly used authentication systems involve the use of text passwords. A usable password must be easy to remember. However, a secure password must be hard to guess. Thus, the challenge presented to usable security is to create authentication schemes that employ "passwords" that are easy for legitimate users to remember and use, but difficult for attackers to obtain or guess. Unfortunately, it just so happens that what is typically memorable is also quite guessable. This problem illustrates the typical trade-off between usability and security. As we increase either security or usability, the other tends to suffer. For example, consider the situation in which a system promotes security by demanding that passwords consist of long strings of random letters, numbers, and special characters. While it would be difficult to guess such passwords, remembering them would be a challenge for most users, and remembering several of them would be nearly impossible without writing them down. Indeed, a large part of the problem with password authentication comes from the cognitive limitations of human memory.

When exploring more usable alternatives to the ubiquitous text password, we might consider some known cognitive propensities. For example, we know that in human memory recognition is typically more effective than recall, and that pictures are more easily remembered than words [28, 29]. Indeed, human ability regarding visual memory is known to be quite strong. Furthermore, there may be a particular aspect of visual memory, like facial recognition, that is especially promising for use in authentication schemes. This is the idea behind *Passfaces*, a commercially available authentication system that has already been implemented by a number of large websites [30]. In Passfaces, users authenticate by correctly recognizing face images previously assigned to them, as they appear among random sets of distractors. A set of faces is what makes up the user's "password", which is called a *graphical password*, since it is based on images rather than text.

In this paper we present a study of this approach to authentication. We implemented a visual recognition scheme similar to Passfaces, and investigated the effect of image type. Earlier work [11] has shown that user choice makes such schemes too guessable, so we assigned users with random passwords. Aware that the theoretical password space typically shown for Passfaces is small (similar to PINs), we instead configured our scheme to have the same space as a 6 character text password using only single-case letters. We then explored the effects of image type by investigating whether face images were superior to images of everyday objects, or images of houses.

The claim that face images are superior stems from psychological research. We therefore review this research (along with other relevant literature) in order to explain the basis of the proposed advantages, and identify some important qualifications. After presenting the results of our studies, we then return to this literature to provide an interpretation and to highlight the complex interplay between psychological theory and its application to knowledge-based authentication.

2. BACKGROUND

2.1 Graphical Passwords

With graphical passwords, users interact with images in order to authenticate. As such, a user "password" is not a word at all. Instead, the "password" arises out of the interaction that takes place between the user and the image(s). De Angeli *et al*. [12] suggest that there are three types of such graphical password schemes: *drawmetric* schemes, *locimetric* schemes, and *cognometric* schemes.

With drawmetric schemes, such as Draw-a-Secret [21], users must recreate figures or shapes by drawing them in order to authenticate. Locimetric schemes present an image that acts as a cue, and users must click on particular regions of the image. Therefore, the proper regions comprise the "password". Originally developed by Wiedenbeck *et al*. [39], *PassPoints* is the most studied such scheme.

In cognometric schemes, users are presented with sets of images and are required to correctly recognize and select a particular image in each set. The "password" is the series of particular images that the user must select. There are many variations on this scheme, but the one most used and studied is *Passfaces* (see Figure 1). In Passfaces, users are given a set of faces to remember. In order to log in, the user must recognize and select each face, as it appears randomly among a grid of distractor faces. A problem with this scheme is that in versions where users are allowed to choose the faces that make up their password, significant biases are found in user selection. Specifically, users predictably choose attractive faces of their own race [11]. From a security standpoint, this vastly increases the chances that attackers will correctly guess user passwords. On the other hand, assigning random images to the user may have negative implications for memorability as users are more likely to remember passwords that they have chosen than ones that have been assigned to them [31]. Since recognition-based graphical password systems have been found highly usable to begin with [5, 37], the gain in security by assigning images to the user is hypothetically deemed worthwhile.

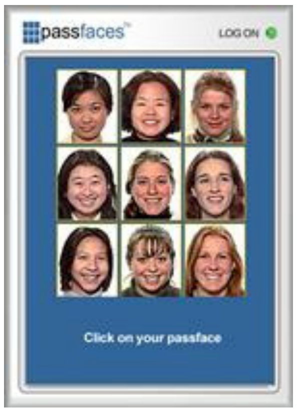


Figure 1. A Passfaces login screen

2.2 Visual Memory

The human mind possesses an exceptional ability to remember images — an ability that has been studied for some time [4, 8, 28, 33, 34]. Moreover, studies have demonstrated that pictures are recalled and recognized more easily than words [3, 27, 29]. Despite this knowledge, the predominant authentication scheme hinges upon the free recall of text passwords.

A study by Bousfield *et al*. [2] showed that recall of words has a positive relationship with the amount of "signs" shown with them during original exposure. Naturally, a picture carries more information than a set of words used to describe it. It is this inherent abundance of information conveyed by an image that is responsible for its increased memorability. Dual-coding theory [28, 29] argues that memory of images is stronger than memory of words because images are more likely than words to be processed *both* visually and verbally. Similarly, Craik and Lockhart [10] proposed the existence of a levels-of-processing effect, whereby the method and depth of processing affects how an experience is stored in memory. Since images present more information to process, and the information may be semantically rich, images are likely to be encoded in more ways, which results in increased availability or access to them.

2.3 Recognition vs. Recall

When comparing traditional text passwords to *cognometric* graphical passwords (used in our studies), it is important to distinguish between two memory processes — recall and recognition. Recall takes place when one thinks back in time and brings to mind information of which one was previously aware. The most distinct recall task, *free recall*, takes place when one is faced with a blank space and asked to fill it in — not unlike entering a password into a text box. Cued recall is slightly different in that the subject is given a hint or clue that assists memory. Recognition occurs when one correctly identifies someone or something that they already know, when it is presented to them at a later time.

Our ability to recognize is generally found to be superior to our ability to recall [23]. This point is commonly illustrated in examples from everyday life. For example, multiple choice questions are frequently easier than essay questions because the correct answer is available for recognition. There is also the tendency to recognize a person's face, but forget their name. While the superiority of recognition is generally acknowledged, there have been a few studies that show little difference, or even the reverse effect. In particular, it appears possible that serial effects allow people to recall items in order very effectively — more so than recognizing them individually [38].

2.4 Face Recognition

The Passfaces scheme is based on the idea that because humans have a special ability for the recognition of faces, face images are the best type to use in cognometric graphical passwords. There is an increasing amount of evidence that there may be regions of the brain dedicated to facial recognition and processing [14, 18, 26]. For example, *Prosopagnosia* (face blindness) is a disorder whereby the ability to recognize people's faces is impaired. Prosopagnostics must rely on other cues to recognize people (voice, clothing style, etc.). While prosopagnostics have been known to occasionally struggle with more general object recognition, they struggle much more with faces. *Visual agnosia* is a more general condition where sufferers are unable to make use or sense of normal visual stimuli. However, facial recognition is preserved in many cases of visual agnosia. This suggests that facial recognition is functionally different than recognition of other visual stimuli [14].

Functional MRI studies have shown that a region in the fusiform gyrus known as the "fusiform face area" (FFA) is activated more strongly when viewing faces as compared to other stimuli. Bilateral FFA activation takes place during both face and object processing, however, when processing faces, right hemisphere FFA activation is a relatively higher correlate. Also active during face processing is the superior temporal sulcus (STS), which appears to process dynamic aspects of facial information, like expression or gaze. There is also an important area in the occipital region termed the "occipital face area" (OFA), which is sensitive to physical features of face stimuli.

Minnebusch *et al*. [26] outline the consensus that all three of these regions are recruited bilaterally (with some right hemisphere bias) during facial processing. When the integrity of this network is disrupted by malfunction in one of the above mentioned areas (as in prosopagnosia), facial processing is impaired. Haxby *et al*. [18] distinguish between recognition of the *invariant aspects* of faces that are responsible for identifying individuals, and the *changeable aspects* of faces such as expression, lip movement, and eye gaze. The invariant aspects are handled by the face-responsive region in the FFA. The changeable aspects are dealt with by the face-responsive region in the STS. These occipitotemporal regions in the extrastriate visual cortex act in concert with neural systems used for other cognitive functions to obtain meaning from faces. Broadly consistent with Minnebusch *et al*. [26], this viewpoint provides specific neurophysiological support for face recognition as a special ability.

There is also evidence from beyond the realm of neurology that facial recognition is distinct from recognition of other visual stimuli. Researchers have studied the vertical inversion of visual stimuli, which disrupts the recognition of faces more than the recognition of other objects. This effect, known as the *face inversion effect*, is often used to support the idea that face recognition is a dedicated process that is different from general object recognition [40]. Other researchers conducted several studies involving the retroactive interference of different types of stimuli in same-different matching tasks with face images [15]. Their main findings provide strong operational evidence for the unique holistic processing of faces. The holistic nature of facial recognition refers to the idea that faces are interpreted using a "Gestalt" impression resulting from the overall set of facial features.

From infancy, we are innately drawn toward faces [22], and over time we all become experts at recognizing and interpreting faces. Diamond and Carey [13] present data that suggests a role of *expertise* in facial recognition. They showed, for example, that dog experts suffer from an inversion effect similar to the facial inversion effect when viewing inverted images of dogs, while non-experts do not. Diamond and Carey do not doubt the neurophysiological support for the uniqueness of face recognition, but show that extensive expertise may also be involved in the high performance of face recognition that is observable. Importantly, we are all predisposed to becoming experts at face recognition as we spend our whole lives exercising that ability.

2.5 Password Space

When comparing the memorability of different password schemes, one must be consistent in the level of security provided. In cryptography, the term "entropy" has been used to define the difficulty in guessing or determining a password, where the randomness of a password determines its strength [7]. The strength of a password scheme can likewise be described in terms of its password space – the number of possible passwords. Ordinarily it is important to distinguish between the *theoretical password space* (all mathematically possible combinations) and the *effective password space* (those combinations more likely to be chosen by users). In our study, however, we assign random passwords, so the effective space is the same as the theoretical space. Clearly, the larger the password space, the more difficult it is for an attacker to guess correctly.

The theoretical space of a text password system involves the number of possible characters and the length of the passwords. For example, imagine a text password system where the only requirement is that the password be exactly six letters long, and is not case sensitive. The number of possible passwords is therefore $26^6 \approx$ 300 million $\approx 2^{28,}$ referred to as a space of 28 bits. For an 8 character password using all 95 characters on a standard US keyboard, the number of possible passwords would be $95^8 \approx$ 6.6 quadrillion $\approx 2^{53}$, or 53 bits; see Table 1 for other examples.

Table 1. Text password criteria and theoretical space

Description	Chars.	Length	Space	Bits
PIN	10	4	1.00×10^4	13
Singlecase	26	6	3.09×10^8	28
Mixed Case	52	6	1.98×10^{10}	34
Full keyboard	95	6	7.35×10^{11}	39
Full keyboard	95	8	6.63×10^{15}	53

For recognition based, cognometric password schemes (like Passfaces), the theoretical password space depends on the number of rows, columns, and panels in a given system. The number of rows and columns determines the number of faces in each panel, and the number of panels determines the number of targets. Table 2 shows these criteria and the resulting theoretical password spaces. For example, in a system that used five rows, five columns, and six panels, the theoretical password space is $\log_2 (5 \text{ x } 5)^6 \approx 28$ bits. Note that this scheme therefore has approximately the same theoretical space as a text password that is six letters long and single case. In order to achieve the theoretical password space afforded by an assigned, mixed case text password that is eight characters long and requires the use of numbers and special characters (53 bits), a cognometric scheme would require twelve rows, eight columns, and eight panels.

Table 2. Cognometric passwords and theoretical space

Rows	Columns	Panels	Space	Bits
3	3	4	6.56×10^3	13
5	5	6	2.44×10^8	28
5	5	8	1.53×10^{11}	37
5	10	5	3.13×10^8	28
8	8	5	1.07×10^9	30
8	10	5	3.28×10^9	32
12	8	8	7.21×10^{15}	53

It is important to note while there is a difference between theoretical and effective password space in typical *text* password schemes, there is no difference in most *cognometric* password schemes. This is because the user typically chooses text passwords, while cognometric password schemes will assign random images in a random order to users in order to avoid the predictability of user-chosen passwords. The result is a larger actual password space for cognometric password schemes, which is a significant security advantage.

3. FIRST STUDY

3.1 Design

The purpose of our first study was to investigate the effect of image type on recognition based graphical passwords similar to Passfaces. We choose three kinds of image: faces, everyday objects, and houses. Faces were chosen as they have been suggested to be the superior type for the scheme. We chose everyday objects because of their strong distinctiveness, and houses because like faces, they show similar structural patterns, with individual differences. We were also interested in the memorability of such schemes when they involve a password space stronger than PINs (typically ≈ 13 bits). We therefore chose a space involving 6 panels of 26 images, thus the same spaces as text passwords with 6 random single-case letters: ≈ 28 bits.

We chose a between-subjects design, with 60 participants randomly assigned to one of the three conditions. We chose the between-subjects approach because we did not want our participants learning three different novel password schemes. The participants were students at our university who were frequent web users. Their age ranged from 18 to 43 (M = 21.10, SD = 4.42). We acknowledge that the general population is much more diverse, but we felt this was adequate for a comparative study, and to establish baseline performance suitable to justify more extensive field studies later based on the results. The studies were approved by our University's ethics committee for psychological research.

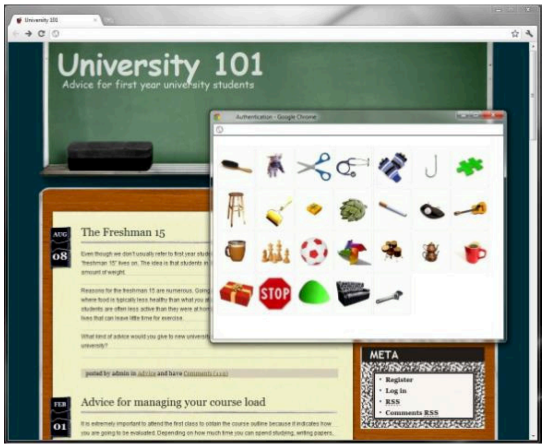

University 101
Advice for first year university students
Authentication - Google Chrome
STOP
The Freshman 15
Advice for managing your course load
META
Register
Log in
RSS
Comments RSS

Figure 2. Logging in to one of the websites.

We implemented the authentication system as a plug-in for several popular open source websites. This approach has been advocated for ecologically valid authentication research [9, 20]. This system (see Figure 2) ensures that we can provide a user experience that differs only in the password scheme involved, and allows us to emphasize real tasks, rather than authentication. We are not aware of any published studies of this kind of recognition based scheme that involved real website use. We created three websites with rich content, and told the participants that we were interested in the whole process of logging in and commenting on articles on the sites, and that the sites involved an unusual password scheme. Each participant used the same websites with the same content. First, we provided a brief in-person introduction to the sites and the password scheme, such as might be done in an office setting. We sent the participants email several times over the course of a week, asking them to log in from home and comment on articles on each of the websites. If passwords were forgotten they could be reset, at the mild cost of additional email interaction and validation: we preferred not to specifically penalize forgetting lest this encourage writing down passwords, and felt the email re-validation was sufficient dissuasion to really spur remembering, and thus had ecological validity. Participants returned to the lab at the end of the week, made final login attempts, and then were provided with a debriefing and compensation.

Samples of images used in our studies are shown in Figure 3. The images of faces used in our studies were obtained with permission from the Face of Tomorrow project [25]. The people in the images used by the commercially available Passfaces authentication scheme [30] are all smiling. Smiling faces are more likely to induce positive affect, which is known to have positive effects on human cognitive performance [1]. Furthermore, Kirita and Endo [24] found that when participants were tasked with sorting faces of a variety of expressions into categories, smiling faces were recognized and sorted fastest. The Face of Tomorrow database features a majority of faces that are smiling, but also includes some neutral expressions. This might be an advantage, when considered in light of the different types of information available when looking at a face identified by Bruce and Young [6]. As facial expression is one of those unique types, it could be argued that it may be easier to distinguish among faces when expression is not uniform. The images in the objects condition were chosen for their distinctiveness, from the stock.xchng website [35], and we used the images in accordance with the published terms and conditions. The images in the houses condition were based on photographs taken by us specifically for use in our study, showing the house frontages presented and framed consistently.

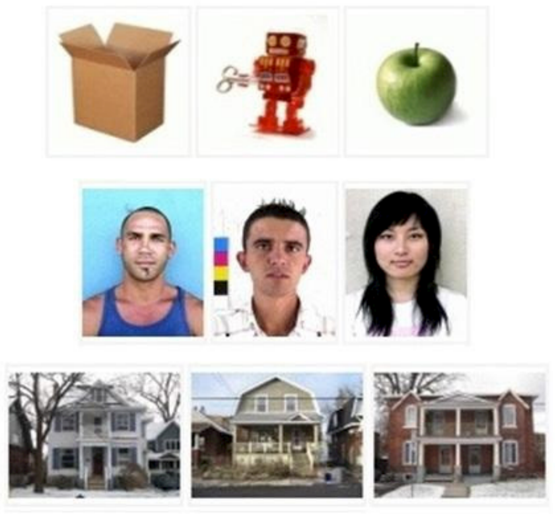


Figure 3. Examples of each image type (objects, faces, houses)

Our system logged all password-related activity on the websites. From these logs we determined the number of login attempts, the time spent in the login process, and whether each attempt was a success or a failure.

3.2 Results

The most important results from the first study had to do with differences between conditions (or image types) in the login success and login times of participants. Login success was measured in two ways. As participants were assigned three graphical passwords, one measure was simply the number of passwords that they remembered after one week. The other measure was the mean amount of time between a participant's first and last successful login – an indication of how long participants remembered their passwords.

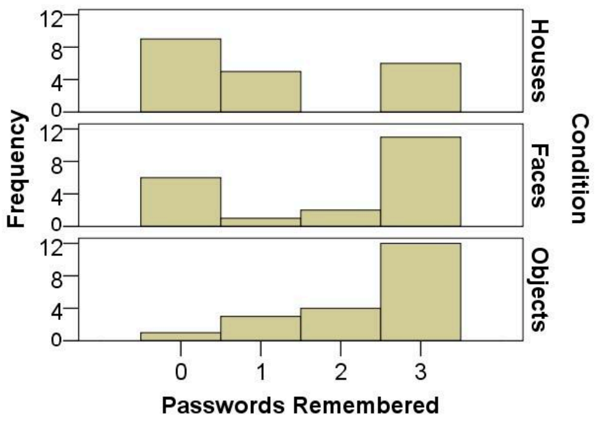


Figure 4. Distributions showing the number of passwords remembered after one week.

After one week, participants in the house images condition were the least able to remember (recognize) the images in their passwords (M = 1.15, SD = 1.31), and participants in the object images condition performed slightly better (M = 2.35, SD = 0.93) than those in the face images condition (M = 1.90, SD = 1.37), as seen in Figure 4. The data was not normally distributed, so we applied the non-parametric Kruskal-Wallis test. This found a significant difference between conditions ($\chi^2(2, N = 60) = 7.62, p = .022$), so we followed up with multiple comparison (Bonferroni-corrected Mann-Whitney U) tests, which found that the only significant difference was between the object and house image conditions (z = -2.85, p = .004).

The other measure used to determine login success with the graphical passwords was called "mean memory time" – the average amount of time between the first and last successful login. Mean memory time varied across conditions in the same way as the other measure of login success, with participants in the house images condition performing the poorest, and those in the object images condition slightly outperforming those in the face images condition, as seen in Table 3. Importantly, although participants using object images remembered their passwords (on average) 31 hours longer than those using face images, the same series of tests found that the only statistically significant difference was between those in the object and house image conditions, (z = -3.54, p < .001).

Table 3. Mean memory time across conditions

	Mean	Median	SD
House images	51.57	14.53	67.19
Face images	103.82	124.66	75.96
Object images	134.79	141.08	60.01

Another important result was the difference between conditions in the average amount of time taken to log in, as seen in Figure 5. After one week, participant login times were by far the slowest for those in the house images condition (M = 83.06, SD = 54.75). Using the same analysis method, participants in the object images condition (M = 31.03, SD = 16.63) logged in *significantly* faster than those in the face images condition (M = 41.45, SD = 14.18) (*z* = -2.53, *p* = .011). Recall that the graphical passwords in this study featured greater security (in password space) than previously tested by adding rows, columns, and panels to the scheme. Interestingly, although the results show that this change resulted in relatively long login times, participants had neutral feelings about how long it took to log in with the scheme.

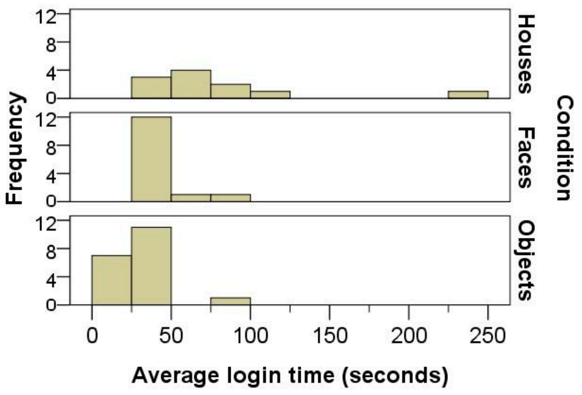


Figure 5. Distributions showing average login times after one week.

3.3 Implications

Although the results of the first study suggested that image type did have a significant effect on the usability of cognometric graphical passwords, there was no evidence that face images were the best image type. In fact, participants in the object images condition showed slightly better memory for their graphical passwords, and showed significantly faster login times.

This first study also suggested that increasing the security of the scheme by adding rows, columns, and panels (increasing its theoretical password space) impacted the usability of the scheme. The cognometric scheme traditionally employs 3 or 4 panels of 9 images and has been shown to be quite usable [37, 5]. Our first study increased password space by using 6 panels of 26 images (a decrease in relative usability), and roughly half of all passwords were forgotten by the end of the one week study.

4. SECOND STUDY

4.1 Design

In order to further explore the implications of our first study, a follow-up study was conducted. This second study was similar to our first study, but with a few changes. First, a within-subjects design was used because our experience with the first study suggested that participants would not find it difficult, and because this approach would also address individual differences between participants, and allow stronger (paired) statistical tests. Participants (n = 20) were assigned two cognometric graphical passwords with face images, and two with object images. The house image condition from the first study was dropped, as participant performance in that condition was quite poor. This two condition within-subjects design allowed for a more direct comparison between image types, where participants could also be asked which image type they preferred, and why. Another difference in the second study was a reduced password space in the graphical passwords that participants were assigned. According to Florencio & Herley [16], a password space equivalent to 20 bits offers sufficient protection from online guessing attacks. As such, we decided to lower password space from 28 bits to 20 bits ($\log_2 (4 \times 4)^5 \approx 20$) by reducing our cognometric scheme to feature five 4x4 panels (see Figure 6).

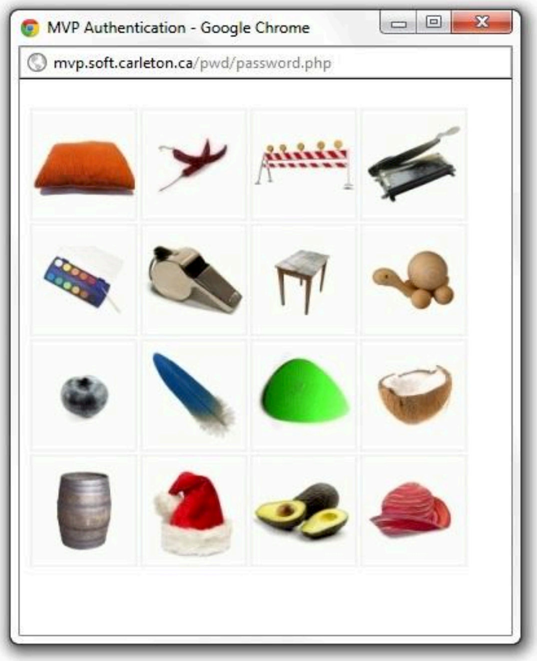

MVP Authentication - Google Chrome
mvp.soft.carleton.ca/pwd/password.php

Figure 6. Panel of object images from the second study.

4.2 Results

The results from the second study were similar to those from the first study. More specifically, a paired t-test found no significant difference between the face (M = 167.80, SD = 51.73) and object (M = 168.50, SD = 42.79) image conditions in mean memory time (t(19) = -0.76, p = 0.46). Indeed, the distributions in Figure 7 show that mean memory time was quite similar for both image types. When comparing login time between conditions, the same test found that participants logged in significantly faster (t(18)= 3.73, p = .002) when using object images (M = 22.55, SD = 10.02) than they did when using face images (M = 35.96, SD = 18.10). Figure 8 shows the distributions for login time across conditions. When asked which of the two image types they preferred in their graphical passwords, 17/20 participants indicated a preference for object images, often citing increased distinctiveness as their reason.

4.3 Implications

Changing the password space of passwords in the scheme appeared have two important effects. First, participant login times in both conditions were much quicker than in the first study. Second, 95% of the object image passwords and 87% of the face image passwords assigned in the second study were remembered for the entire week. While it should not be surprising that decreasing the security of the scheme would result in an increase in usability, such a vast increase in usability is enough to render the scheme a more practical alternative to text passwords.

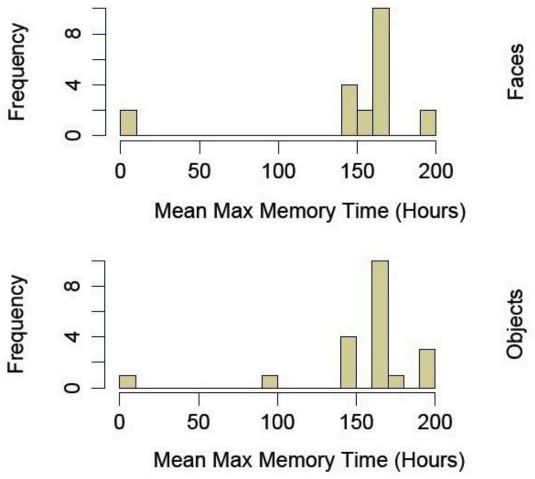


Figure 7. Password memory time in the second study.

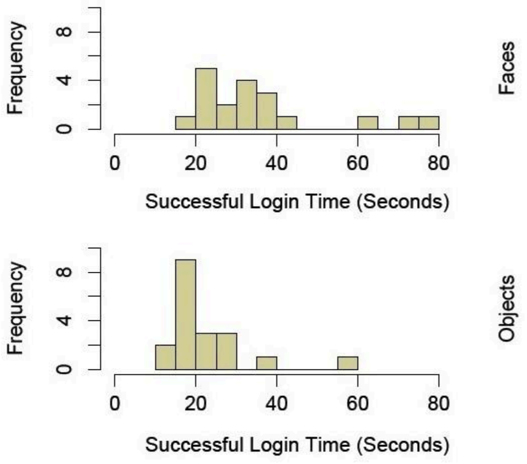


Figure 8. Password login time in the second study.

5. DISCUSSION

Contrary to what is suggested by Passfaces [30], the results of our studies suggest that face images may not be the best type of images to use in cognometric graphical passwords. While participants in the first study certainly performed better with face images than they did with house images, any evidence of superiority over the object images condition was completely absent. The special way in which people process and thus recognize the faces of others, along with the fact that we all bear a certain expertise in doing so, is certainly advantageous when using a recognition-based authentication system featuring face images. However, it is possible that different advantages afforded by other image types are better suited to the task.

Indeed, the object images used in our study have their own set of advantages for memory and recognition that are less available when looking at images of unfamiliar faces. For one, the images in the object image condition look vastly different from one another. Each object has its own shape and size. Perhaps most important is that these colourful, high quality images with their uniform white backgrounds were chosen for the object images condition. That the colours were so bright and so different for each image may have been of great service to participants in the object images condition. This stands in contrast to the images used in the face images condition, which featured much less variation in shape, size, and colour. The second way that each image in the object images condition is unique has more to do with the vast semantic difference between images. The object images condition featured images of tools, toys, food, flowers, stationery items, furniture, and more. Although faces of varying age, race, gender, expression, etc. bear some semantic difference to the viewer, these differences are dwarfed by the semantic difference between any two categories of objects in the object images condition, and a given object image password was likely to feature images from several such categories.

One may speculate that the greater visual and semantic differences between images in the object images condition allowed participants to find and recognize specific images more quickly. These increased visual and semantic differences may have resulted in a more distinct experience for the user when looking at each image. Gallo, Meadow, Johnson, and Foster [17] discuss *distinctiveness* in terms of the complexity and uniqueness of the perceptual features in the stimulus, claiming that "distinctive features potentially help subjects to differentiate picture memories, and also provide more features that can subsequently be recollected" (p. 1097). It is the complexity of the differences between random objects that renders their perceptual features so unique. In other words, "distinctiveness can be influenced by the uniqueness of semantic or conceptual features in memory, holding the perceptual input relatively constant" (p. 1109). The images in the object images condition had greater distinctiveness than the images in the other two conditions, and yet their presentation was the most constant. The uniform presentation of the object images, each one centered and isolated on a white background, is not to be underestimated. Similar to how Hunt [19] describes distinctiveness as difference in the context of similarity (see page 18), Gallo et al. [17] stipulate that certain perceptual constants are upheld. The uniform, almost exhibit-like presentation of the objects (along with their categorization *as* objects) created this constant context in which the vast perceptual and semantic differences between objects were more obvious.

Once again of importance is Craik and Lockhart's levels-of-processing framework [10], and more specifically how it relates to distinctiveness. Recall that, according to the levels-of-processing framework, the method and depth of processing affects how an experience is stored in memory. Gallo et al. [17] argue that "the levels of processing effect on memory is based on recollective distinctiveness" (p. 1109), whereby the distinctiveness of an experience is what determines our ability to remember it. Furthermore, that "uniqueness or distinctiveness within a level of processing can influence performance [is] consistent with many other studies highlighting the importance of distinctiveness for recall and recognition". Gallo et al. emphasize that it is the uniqueness of the stimulus during encoding and retrieval that plays such an important role in memory. The authors argue that "the most effective way to encode items for subsequent recall or recognition is to associate each item with information from pre-existing knowledge that can later provide a large number of unique features to retrieve". The relevance of this argument is not to be overlooked. When comparing our ability to remember faces of people we do not know to our ability to remember objects that most of us know a great deal about, there is a great difference in the amount of pre-existing knowledge available for association.

Of the different types of information available when looking at a picture of someone's face, identity-specific information is said to contain all the information that cannot be derived by simply looking at the picture [6]. It could be argued that the most important aspects of knowledge pertaining to the subject matter of such a picture — a person — is not available when simply looking at it. This is not typically the case when looking at an image of an orange, for example. Most people have had all kinds of first (and second) hand experiences with oranges through which they have learned the information most pertinent to them. That is to say, most people know more about "everyday" objects than they know about people they have never seen or heard of before. As a result, there is a greater opportunity for people to associate images of those objects with their own knowledge and/or experience.

Another important difference between image types had more to do with the strategies reported by participants to help themselves remember their passwords. In the house and face image conditions, participants often reported that they created some brief verbalization of each image in their password. For example, they would come up with names such as "sporty blonde" for a certain face image or "the barn house" for a certain house image. This labeling process seemed less important for those in the object images condition, perhaps because each object already had its own "name". This could be said to relate to the previous point about association — that because there are less unknowns to the subject matter in the object images, there is greater opportunity for associations to be made. Regardless, it seemed that the verbalization of images (particularly in the house and face image conditions) was used by participants to assist the recall of the images in their password. Often, participants would attempt to recall the verbalizations of the images in their graphical password before any attempt to recognize them on the screen. In instances where recall was unavailable, participants would then resort to relying purely upon recognition as they slowly scanned through each panel of images. It was during this pure recognition task that interference from distractor images (that may have been relevant for another password) seemed to be the strongest, contributing to slower login times.

The difference between conditions in how long it took participants to log in was an important test, as users generally do not want to spend large amounts of time in the authentication process. If graphical passwords are to be seriously considered as a usable alternative to text passwords, they will have to satisfy the user's expectation for what a reasonable amount of time spent in the authentication process is. Post-test questionnaire feedback from the first study showed that participants in every condition felt that it took a bit too long to authenticate with the recognition based password scheme. Although time taken to successfully log in is more of a usability measure than any measure of memorability, the more time participants spent looking for the images in their password, the greater the opportunity for distractor interference would become. When participants took longer to log in, it was partly because they had to look through more images before locating and recognizing those from their password set. Participants reported that this interference from distractor images that were not in their password had a disruptive effect on their ability to recognize the correct images.

The studies have two implications for visual memory in general. First, expertise with the subject matter of the visual stimulus is advantageous for the memorability of that stimulus. Second, the more distinct that the experiences resulting from a visual stimulus are, the more distinctly it is processed, which facilitates later memory. Interestingly, it could be argued that these two ideas are related, as the benefits of expertise may be driven by an enhanced ability to make important distinctions among specific types of subject matter.

6. CONCLUSIONS

This paper has reported on two studies that explored the effect of image type in recognition-based graphical passwords. It has been suggested that face images are the ideal image type, but we found no evidence to support that claim. In fact, the results from our studies suggest that distinctive object images perform at least as well as face images when it comes to memorability, and are superior in terms of login time. The claim that face images have superior memorability stems from psychological research suggesting a "special" ability to process and remember faces. We reviewed this literature, along with related research on expertise and distinctiveness, and conclude that although we may have a *special* ability to process and memorize faces, this does not necessarily lead to a *superior* ability (in the context of knowledge-based authentication).

The scheme in our first study featured a password space of 28 bits. As only half of the participants in the most usable condition of that study remembered their passwords, we conclude that the scheme is less practical as an alternative to text passwords. When we lowered password space to 20 bits in the second study, 92% of the passwords were remembered for the one week duration of the study. This shows more promise. User choice in recognition-based passwords has been identified as problematic in earlier research, who suggested that random assigned passwords would be preferable [11]. Our studies employed this strategy, and the memorability shown in the second study suggests that this approach is usable in practice, and has the advantage that the full theoretical password space is used, with no particular password being more common than any other.

7. REFERENCES

[1] Ashby, G., Isen, A., & Turken, A. (1999). A neuropsychological theory of positive affect and its influence on cognition. Psychological Review, 106(3), 529-550.

[2] Bousfield, W. A., Esterson, J., & Whitmarsh, G. A. (1957). The effects of concomitant colored and uncolored pictorial representations on the learning of stimulus words. Journal of Applied Psychology, 41, 165-168.

[3] Bower, G. H. (1972). Mental imagery and associative learning. In L. Gregg (Ed.), Cognition in Learning and Memory (51-88). New York: Wiley.

[4] Bower, G. H., Karlin, M. B., & Dueck, A. (1975). Comprehension and Memory for Pictures. Memory and Cognition, 2, 216-220.

[5] Brostoff, S., & Sasse, M. (2000). Are Passfaces more usable than passwords? A field trial investigation. British Human-Computer Interaction Conference (HCI), September 2000.

[6] Bruce, V., & Young, A. (1986). Understanding face recognition. British Journal of Psychology, 77(3), 305-327.

[7] Burr, W., Dodson, D., Polk, W. (2006). Electronic Authentication Guideline: Recommendations of the National Institute of Standards and Technology. NIST Special Publication 800-63.

[8] Calkins, M. W. (1898) Short studies in Memory and Association from the Wellesley College Laboratory. Psychological Review, 5, 451-462.

[9] Chiasson, S., Deschamps, C., Stobert, E., Hlywa, M., Freitas Machado, B., Chan, G., & Biddle, R. (2010). The MVP Web-based Authentication Framework, Technical Report TR-10-19, School of Computer Science, Carleton University, Ottawa, Canada.

[10] Craik, F., & Lockhart, R.S. (1972). Levels of processing: A framework for memory research. Journal of Verbal Learning & Verbal Behavior, 11(6), 671-84.

[11] Davis, D., Monrose, F., & Reiter, M. K. (2004). On User Choice in Graphical Password Schemes. Proc. of the 13th USENIX Security Symp., 151-164.

[12] De Angeli, A., Coventry, L., Johnson, G., & Renaud, K. (2005). Is a picture really worth a thousand words? Exploring the feasibility of graphical authentication systems. International Journal of Human-Computer Studies, 63(1-2), 128-152.

[13] Diamond, R., & Carey, S. (1986). Why faces are and are not special: an effect of expertise, Journal of Experimental Psychology: General, 115, 107-117.

[14] Farah, M.J. (1996) Is face recognition "special"? Evidence from neuropsychology. Behavioural Brain Research, 76(1-2), 181-189.

[15] Farah, M.J., Wilson, K. D., Drain, M., & Tanaka, J. (1998). What is "special" about face perception? Psychological Review, 105(3), 482-498.

[16] Florencio, D., & Herley, C. (2010). Where Do Security Policies Come From? In Proc. of the Sixth Symp. on Usable Privacy and Security (SOUPS). New York, USA: ACM.

[17] Gallo, D. A., Meadow, N. G., Johnson, E. L., & Foster, K. T. (2008). Deep levels of processing elecit a distinctiveness heuristic: Evidence from the criterial recollection task. Journal of Memory and Language, 58, 1095-1111.

[18] Haxby, J. V., Hoffman, E. A., & Gobbini, M. I. (2000). The distributed human neural system for face perception. Trends in Cognitive Science, 4, 223-233.

[19] Hunt, R. R. (2006). The Concept of Distinctiveness in Memory Research. In R. Hunt & J. Worthen (Eds.), Distinctiveness and Memory (3-26). New York: Oxford University Press.

[20] Inglesant, P., Sasse, M. A. (2010). Studying Password Use in the Wild: Practical Problems and Possible Solutions. SOUPS 2010: Workshop on Usable Security Experiment Reports.

[21] Jermyn, I., Mayer, A., Monrose, F., Reiter, M. K., & Rubin, A. D. (1999). The design and analysis of graphical passwords. Proc. of the 8th conference on USENIX Security Symp., p.1-14, August 23-26, 1999, Washington, D.C.

[22] Johnson, M., Dziurawiec, S., Ellis, H., & Morton, J. (1991). Newborns' preferential tracking of face-like stimuli and its subsequent decline. Cognition, 40(1-2), 1-19.

[23] Kausler, D. H. (1974). Psychology of Verbal Learning and Memory. New York: Academic Press

[24] Kirita, T., & Endo, M. (1995). Happy face advantage in recognizing facial expressions. Acta Psychologica, 89(2), 149-163.

[25] Mike, M. (2010) the Face of Tomorrow Project. Available online at http://www.faceoftomorrow.com/

[26] Minnebusch, D.A., Suchan, B., Koster, O., & Daum, I. (2009). A bilateral occipitotemporal network mediates face perception. Behavioural Brain Research, 198(1), 179-185.

[27] Nelson, D. L., Reed, U. S., & Walling, J. R. (1977) Pictorial Superiority Effect. Journal of Experimental Psychology: Human Learning and Memory, 3, 485-497.

[28] Paivio, A., Rogers, T. B., & Smythe, P. C. (1968) Why Are Pictures Easier to Recall Than Words? Psychonomic Science, 11, 137-138.

[29] Paivio, A. (1973). Picture superiority in free recall: Imagery or dual coding? Cognitive Psychology. 5(2), 176-206.

[30] Passfaces Corporation, "The science behind Passfaces," White paper, http://www.passfaces.com/enterprise/resources/white_papers.htm, accessed November 2009.

[31] Renaud, K. (2009). On user involvement in production of images used in visual authentication. Journal of Visual Languages and Computing, 20(1), 1-15.

[32] Sasse, M. A., & Flechais, I. (2005). Usable Security. Why do we need it? How do we get it? In Cranor, & S. Garfinkel (Eds.), Security and Usability (pp. 13-30). Sebastopol, CA: O'Reilly Media, Inc.

[33] Shepard, R. N. (1967). Recognition Memory for Words, Sentences, and Pictures. Journal of Verbal Learnings and Verbal Behavior, 6, 156-163.

[34] Standing, L. (1973). Learning 10,000 Pictures. Quarterly Journal of Experimental Psychology, 25, 207-222.

[35] Stock.xchang (2010) The leading free stock photography website. Available online at http://sxc.hu/

[36] Thorpe, J., & Van Oorschot, P. C. (2007). Human seeded attacks and exploiting hot-spots in graphical passwords, in 16th USENIX Security Symp., 2007.

[37] Valentine, T. (1999). And evaluation of the Passface personal authentication system. Goldsmiths College University of London, Tech. Rep., February 1999.

[38] Watkins, M. J. (1974). When is recall spectacularly higher than recognition? Journal of Experimental Psychology, 102(1), 161-163.

[39] Wiedenbeck, S., Waters, J., Birget, J.C., Brodskiy, A., & Memon, N. (2005). PassPoints: Design and longitudinal evaluation of a graphical password system. International Journal of Human-Computer Studies, 63(1-2), 102-127.

[40] Zhao, W., Chellappa, R., Phillips, P. J., & Rosenfeld, A. (2003). Face Recognition: A Literature Survey. ACM Computing Surveys, 35(4), 399-458.

PhorceField: A Phish-Proof Password Ceremony

Michael Hart Claude Castille[†] Manoj Harpalani
Jonathan Toohill[†] Rob Johnson

{mhart,mharpalani,rob}@cs.sunysb.edu
{claude.castille,stonethumb}@gmail.com[†]

Stony Brook University

ABSTRACT

Many widely deployed phishing defense schemes, such as SiteKey, use client-side secrets to help users confirm that they are visiting the correct website before entering their passwords. Unfortunately, studies have demonstrated that up to 92% of users can be convinced to ignore missing client-side secrets and enter their passwords into phishing pages. However, since client-side secrets have already achieved industry acceptance, they are an attractive building block for creating better phishing defenses. We present PhorceField, a phishing resistant password ceremony that combines client-side secrets and graphical passwords in a novel way that provides phishing resistance that neither achieves on its own. PhorceField enables users to login easily, but forces phishers to present victims with a fundamentally unfamiliar and onerous user interface. Victims that try to use the phisher's interface to enter their password find the task so difficult that they give up without revealing their password. We have evaluated PhorceField's phishing resistance in a user study in which 21 participants used PhorceField for a week and were then subjected to a simulated phishing attack. On average, participants were only able to reveal 20% of the entropy in their password, and none of them revealed their entire password. This is a substantial improvement over previous research that demonstrated that 92% of users would reveal their entire password to a phisher, even if important security indicators were missing[27].

PhorceField is easy to deploy in sites that already use client-side secrets for phishing defense – it requires no client-side software and can be implemented entirely in javascript. Banks and other high value websites could therefore deploy it as a drop-in replacement for existing defenses, or deploy it on an "opt-in" basis, as Google has done with its phone-based "2-step verification" system.

1. INTRODUCTION

High-value web services, such as online banking, have deployed client-side secrets, like those used in SiteKey, as part of their phishing defense systems. These schemes are intended to help users identify a site as valid so they may be confident they are entering their credentials into the proper prompt. Client-side secrets have achieved tremendous industry adoption. For example, Bank of America has deployed its SiteKey system to over 15 million online banking customers.

Phishing is a major form of online fraud. Phishing attacks require little technical skill and are easy to launch. Furthermore, since users often re-use passwords on multiple sites, passwords gained in a phishing attack against one website are often re-usable on other sites. Over 3.6 million people in the U.S. lost money to phishing attacks in 2007, losing over 3 billion USD[15]. Since then, phishing attacks have grown much more sophisticated. For example, Google recently claimed that it uncovered a coordinated set of spear-phishing attacks against high-profile and high-value targets[25]. These attacks use emails that appear to come from their victims' friends and co-workers and contain plausible details to trick victim's into lowering their guard. It is unreasonable to expect users to be able to detect such attacks.

Despite the widespread deployment of client side secrets, there is strong evidence that they provide poor phishing protection: 92% of participants in one study ignored missing security indicators, including their client-side secret. Current client-side secret systems depend on the user to verify the presence of the secret. A phisher can therefore circumvent the defense mechanism by tricking users into ignoring the missing secret. To solve this problem, we need to force users to verify the presence of the client-side secret before entering their password.

In this paper, we present PhorceField: a novel integration of client-side secrets and graphical passwords that better utilizes existing client-side secrets while alleviating the users of the responsibility for creating, maintaining and supplying passwords. With PhorceField, a legitimate password prompt has access to a secret set of images that enables it to create an easy-to-use password prompt. Phishers do not have access to the secret images and hence must present victims with a fundamentally different and more difficult interface. Users cannot ignore the differences, as with previous schemes

Metric	SiteKey	PhorceField
Percent users revealing their entire password	92%	**0%**
Average amount of password revealed	92%	**20%**

Table 1: Success rate of phishing attacks on PhorceField and SiteKey. The SiteKey statistics are derived from Schechter, et al. [27], so may not be directly comparable to our results. Since SiteKey users reveal all-or-nothing of their password, the average amount of password entropy revealed is the same as the percentage of subjects who revealed their password. In our PhorceField study, all users were assigned passwords with 70 bits of entropy, and they revealed an average of 13.8 bits of information about their password.

such as SiteKey, and hence cannot slip into dangerous click-whirr behaviors[18]. Even if users do attempt to interact with the phisher's page, though, the phishing page requires so much effort that the victims give up before they can reveal their password.

PhorceField exploits well-known strengths and weaknesses of human memory[3, 9]. During a normal PhorceField login, users must perform an image recognition task, which is relatively easy for humans. During a phishing attack, though, victims must perform an image recall task, which is substantially harder. Furthermore, PhorceField is designed so that victims will experience memory interference during a phishing attack, causing additional frustration and error. To our knowledge, PhorceField is the first password ceremony to take advantage of these properties of human memory.

PhorceField is easy to use. The PhorceField user experience is identical to a cognometric graphical password[10], which have been shown to be usable and to have memorable passwords. Thus, this paper focuses only on PhorceField's security against phishing attacks.

We have conducted a user study to evaluate PhorceField's ability to resist phishing attacks. Participants used PhorceField for one week and were then presented with a simulated phishing attack on their PhorceField password. We made special effort to ensure the ecological validity of our study and to avoid participant bias. For example, participants worked on their own computers in their normal environments, and we told participants that the study was focused on "usability" instead of security.

As the user study results summarized in Table 1 show, PhorceField users presented with a phishing attack give up before they are able to reveal their entire password. Participants in our study revealed, on average, only 13.8 bits of information (out of 70 bits of entropy) about their password, and no participant revealed his entire password. This is a substantial improvement over recent results on SiteKey, the current industry standard for preventing password phishing, that show that 92% of SiteKey users will reveal their password to a phisher[27].

PhorceField combines two existing techniques – client-side secrets (e.g. secure HTTP cookies) and graphical passwords – in a novel way to achieve a level of phishing-resistance that neither technology achieves on its own. Our results demonstrate that previous schemes based on client-side secrets, such as SiteKey[4] and Dynamic Security Skins[11], are not extracting the full benefit of the client-side secret. PhorceField could serve as a drop-in replacement for the login ceremony of either of these schemes, substantially improving their phishing-resistance.

In summary, PhorceField demonstrates three novel techniques for creating phishing-resistant password ceremonies:

- It forces phishers to present a fundamentally different interface to their victims. This gives users a better chance to detect the attack. With previous schemes, phishers can create interfaces that differ only superficially from legitimate ones.
- It forces phishing victims to complete a much more difficult task. Even if a victim is fooled by the phisher's attack, she is unlikely to succeed in communicating her password to the phisher.
- It exploits strengths and weaknesses of the human memory system to make phishing attacks difficult for their victims. Victims of a phishing attack experience memory interference while looking through hundreds of similar images, causing frustration and error.

We make the following additional contributions:

- We present results from a user study demonstrating that PhorceField successfully protected all participants against a simulated phishing attack.
- We show how PhorceField can easily integrate into existing anti-phishing solutions, such as SiteKey and DSS, without introducing any new hardware or software requirements.

2. BACKGROUND

Phishers steal user credentials by tricking victims into revealing private information, such as passwords. In this section, we review (1) why phishing is not solved by existing technologies, such as SSL or malware defenses, (2) that some proposed phishing solutions offer some protection but do not address the core problem of password disclosure, and (3) that previous solutions targeted specifically at preventing password disclosure are not effective.

Non-solutions.. Phishing is a separate problem from malware (such as key-loggers), click-jacking, and other techniques that attackers may use to steal user secrets. Even if a user's computer is free of malware and has effective defenses against click-jacking, a phisher may still trick the user into divulging his password. Phishing cannot be solved solely with cryptography. SSL and PAKE protocols[5] cannot protect the user's password if she types it into a phisher's website. Password managers and one-time passwords do not prevent phishing, either. A phisher can defeat a password manager by convincing victims to disable it and type in their master password. Given the gullibility of users[27] and the usability problems with password managers[6], the ruse is likely to succeed. One-time password generators, whether implemented as a special-purpose token, software on a cellphone, or delivered to the user's phone via SMS, all require users to manually copy the one-time password from the device to their computer. Phishers can trick users into entering the password into their page instead of the user's intended website, as was done in a real phishing attack against Citibank[19].

Therefore, we need separate defenses to help users avoid entering passwords into malicious prompts.

Anti-phishing toolbars[23, 13], spam filters[26], phishing site blacklists and white-lists[31], and other reactionary mechanisms for fighting phishing offer some protection[7, 35, 1, 2], but phishers have developed many techniques for evading these defenses[22], so millions of users fall victim to phishing each year[15].

Two-factor authentication can mitigate the damage of a stolen password, but it does not protect the password itself. PhorceField can complement a second authentication factor, but it can also protect a user's password even when the password is the sole authentication factor, as in our prototype.

The solutions above fail to prevent phishing because they do not address the core weakness exploited by phishers: Users cannot determine whether a password prompt is legitimate or malicious.

Prior secure password prompt proposals.. Security researchers have made several attempts to create password schemes that will help users distinguish legitimate prompts from malicious ones.

Secure attention keys allow users to summon a trusted password prompt by pressing a special sequence of keys – typically Ctrl-Alt-Del. Secure attention keys do not prevent phishing since phishers can easily trick users to skip the special key sequence.

Dynamic Security Skins (DSS)[11] and SiteKey[4] are conceptually similar schemes that use a client-side secret to help users recognize legitimate password prompts. In these schemes, a secret image known to the user is stored on the user's device. This image is presented to the user as part of the standard password prompt. The user is supposed to verify the image is present before entering her password. Figure 1 shows the protocol for logging in using these schemes. SiteKey and DSS differ in the location of the image: SiteKey displays it above the password entry field, DSS displays it behind the field.

Unfortunately, 92% of SiteKey users will ignore a missing image and enter their password anyway[27]. We could find no published user study evaluating phishing attacks against Dynamic Security Skins, but it also depends on user vigilance for security and so is likely to offer little protection against phishing.

Like SiteKey and DSS, PhorceField leverages a client-side secret to prevent password phishing. Any system that prevents users from communicating passwords to

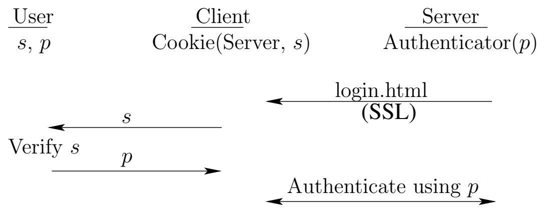


Figure 1: A generalized variant of the SiteKey and Dynamic Security Skins login ceremonies. The user's secret image, s is stored on her machine and is accessible only to legitimate password prompts, i.e. prompts from "Server" in the case of SiteKey, or prompts presented by the DSS plug-in in the case of Dynamic Security Skins. Users are supposed to verify that s is displayed in the prompt before entering their password. Once the user has entered her password, any password-based authentication protocol can be used to authenticate the user to the remote server.

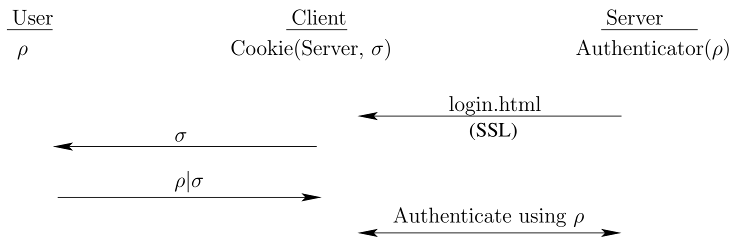
phishers must use either a client-side secret or some client-side peripheral that is not available to the phisher, since otherwise phishers would be able to emulate the password prompt with sufficient fidelity to fool many users. Thus, the only question is: how can we extract the most benefit from this client-side secret? PhorceField achieves a far greater level of phishing resistance than previous schemes, such as SiteKey and DSS, without imposing any additional setup or portability costs.

3. PhorceField

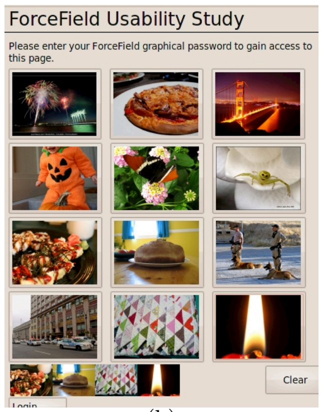
Our goal is to create a password ceremony such that attackers cannot present a usable password prompt to their victims. Therefore, legitimate prompts must have access to some secret that is not known to attackers. Furthermore, passwords cannot consist of letters, numbers, etc., since phishers can present easy interfaces for inputting those symbols. PhorceField satisfies the above two requirements by using a cognometric graphical password scheme and by storing the graphical password images securely on the user's device. Figure 2(a) shows the PhorceField login ceremony and Figure 2(b) shows an instance of a PhorceField password prompt.

Cognometric graphical passwords present users with a set, σ, of images and users log in by clicking on the images in a certain sequence or by clicking on a certain subset of images. Prior work by Moncur, et al. has shown that cognometric passwords are usable and memorable[21]. We refer to the user's password as ρ. In PhorceField, the set σ must be drawn from a much larger set, Σ. Thus, the user's password is a word in the language Σ^* but, since the prompt already has access to σ, the user only needs to communicate $\rho|\sigma$, which may only constitute 10-20 bits of information. A phisher that does not know σ, though, must trick the user into revealing ρ, which may require the user to communicate hundreds of bits of information, making the task much harder.

For security, Σ should be as large as possible and, for usability, σ should be small. For our prototype imple-



(a)


ForceField Usability Study

Please enter your ForceField graphical password to gain access to this page.

(b)

Figure 2: (a) The PhorceField ceremony. The user's secret image set, σ, is stored on her device and inaccessible to phishers. (b) An example PhorceField password prompt using Creative Commons licensed images from the Flickr photo-sharing site.

mentation, we chose $|\sigma| = 12$, since this makes it easy to enter passwords on cell phones and other mobile devices with the standard phone keys 0-9, "*", and "#", and Σ consisted of 188218 creative-commons licensed images collected from the Flickr photo service[34]. There are over 100 million such images on Flickr, so our prototype could be easily scaled up for real-world deployment[30]. We collected images by searching for 193 different concrete nouns, such as "cow", "flowers", "sky", and "tree". We constructed each participant's σ by selecting 12 concrete nouns and then selecting an image from each noun's image set. Passwords in our implementation consist of a sequence of 4 distinct images (order matters). Our system randomly generated passwords for participants. This yields an entropy of 181 bits for σ, 70.0 bits for ρ, and 13.5 bits for $\rho|\sigma$. Our scraper also downloaded the description, tags and titles for each image, which we use later to develop the phishing attack interface in our user study.

Each user's σ must be stored on her device. Our implementation installed σ on participants' computers when they enrolled in our user study. We did not provide any mechanism for initializing σ on a second computer, so participants could only log in from the computer they used to register in the study. A full implementation could use a conditioned-safe ceremony[18] to initialize other computers with σ.

4. PHISHING ATTACKS ON PHORCEFIELD

We argue in this section that phisher's only have two possible strategies for extracting ρ from their victims: brute force attacks and search attacks.

Brute force attacks. A phisher that does not know σ may attempt to learn ρ by presenting the user with an invalid password prompt. If the invalid prompt contains some images from the user's set, σ, then the user may click on them. Note that in many cases, the user may refuse to interact with a prompt unless it contains exactly the images in her σ but, for this analysis, we pessimistically assume that the user will select all the images in her ρ, even for a badly malformed prompt. By performing this attack repeatedly with different images each time, the attacker may eventually recover all of ρ. The attacker can reduce the number of repetitions required to complete the attack by placing more images into each prompt. However, if the number of pictures in the invalid prompt is too large, then users will give up in frustration, and the attacker will learn nothing. If we assume that users will give up if asked to examine more than m pictures in one prompt, then the probability that a phisher can recover σ after repeating the attack r times is $\binom{mr}{|\sigma|}/\binom{|\Sigma|}{|\sigma|}$. For example, our user study described in the next Section used $|\sigma| = 12$ and $|\Sigma| \approx 2 * 10^5$. From the results of our study, we can estimate that $m \leq 3500$ and $r \leq 10$. With these parameters, the probability a phisher can recover σ with this attack is less than 2^{-30}.

Search attacks. Since brute force attacks will not work, a phisher must trick a user into communicating the images in her ρ to the phisher. This is a search problem: the phisher can iteratively make adaptive oracle queries to the victim in order to narrow down the search space for ρ. Note that these queries may be made during a single session or across multiple sessions with the user. PhorceField resists search attacks in several ways.

With PhorceField, attackers must present victims an interface that is substantially different from a legitimate PhorceField password prompt. Victims of phishing attacks therefore cannot slip into a "click-whirr" response mode[18]; they must actively evaluate the situation, giving them a much better chance of detecting the attack.

PhorceField trains users to distinguish their password images from a small set of distractor images – they do not need to remember every detail of their password images. A phisher, however, must coax enough information out of his victims in order to identify their images within a set of thousands or millions of candidate images. In many cases, users will be unable to provide more than the most prominent features of their password images, leaving the phisher with potentially thousands of possible matches. He can try to get the victim to sift through the matches, but our user study shows that victims will often give up before finding their password images.

We constructed the database of images used in our PhorceField prototype so that logging in is easy but cooperating with a phisher is difficult. Recall that Σ contains 188218 images representing 193 different concrete nouns, or about 975 images per noun. Constructing the database in this way causes users to suffer from memory interference while looking through candidate matches for their password images[9, 3]. As the phisher presents the user with more and more candidate images that are similar to the victim's target image, the victim loses her ability to precisely identify her target image because it becomes blurred together with the candidates. This leads to errors, as victims may accidentally select a wrong image, and frustration within the victims, causing them to quit cooperating with the phisher before they succeed in finding their password images.

The preceding arguments show that brute force attacks are impractical and that search attacks, no matter how clever, will require significantly more time and mental effort from victims than is required to log in. However, we cannot analytically determine how users will react when presented with a search attack on their password; for that, we need a user study.

5. USER STUDY

The purpose of our user study is to measure the success of phishing attacks against PhorceField passwords. This study does not attempt to measure the usability of the PhorceField prompt or the memorability of graphical passwords – those topics have been explored elsewhere[10]. Our study includes numerous conservative design decisions, so we believe the results of this study represent an upper bound on the success probability a phisher can achieve with this attack.

Participants in our study were told that we were conducting an experiment to evaluate the usability of graphical passwords. After consenting to the study, participants were shown their set σ and password ρ and required to practice entering their password five times. They then downloaded a Firefox plug-in that randomly prompted them to enter their PhorceField password up to four times per day. By prompting participants randomly four times per day, we ensured that participants were familiar with their passwords at the time of our simulated phishing attack.

Participants were told that the study would last two weeks and that they would receive \$20 if they entered their password at least 14 times over the entire study period, \$10 if they entered their password between 7 and 13 times, and no compensation if they entered their password fewer than 7 times. After about one week, they received an email from us indicating that we had

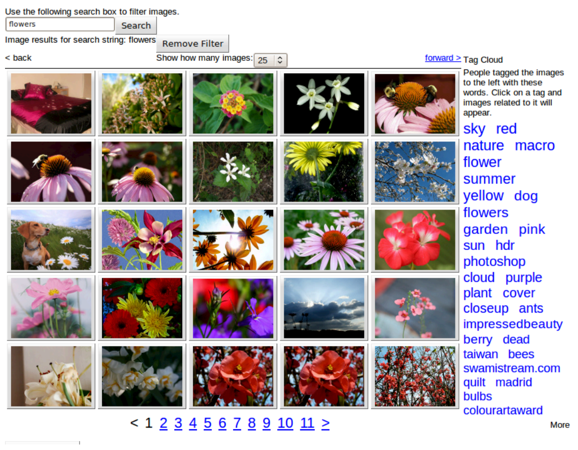


Figure 3: Our phishing attack website. Users could search for images using the search box at the top of the page and could click on tags in the tag cloud on the right-hand side.

lost their password and requesting them to visit the study website to help us recover it. We then measured how much information they were able to divulge about their σ and ρ. Participants were then presented with a debriefing questionnaire.

We were careful to avoid priming the participants about security or phishing in particular. Subjects were told that the study was about usability, not security. All study materials used the name "ForceField" instead of "PhorceField". The attack email clearly came from us and directed the users to the same website that they visited to sign up for the study. Thus our attack was missing all the cues of a real phishing attack. Although subjects knew that the study was about passwords and therefore may have had some security priming, prior work has found that priming does not cause participants to behave more cautiously[27]. Based on all these facts, we believe that our results are an upper bound on the success rate a real attacker would experience with this attack.

Figure 3 shows a screen-shot of the password recovery page. Participants could use the search box to enter queries and could click on words in the tag cloud on the right-hand side of the page. As is demonstrated in the screen-shot, the search results were quite good because the Flickr photos were so well tagged. The interface was modeled after the Google image search interface at the time of the study and, like Google, never indicated the number of images that participants had to search through, since we felt that would only discourage them.

We conducted a pilot study to determine the number of participants needed to estimate the mean number of σ images revealed by a user. During our pilot study, thirteen subjects used our graphical password prompt for seven days and received the phishing attack page under the guise that the experimenters lost their password. The subjects revealed on average 0.35 images in their σ (SD = 0.60). Therefore, we require 20.7 subjects to determine the average number of σ images revealed in a phishing attack with a 99% confidence interval ± 0.34 images[12].

6. RESULTS

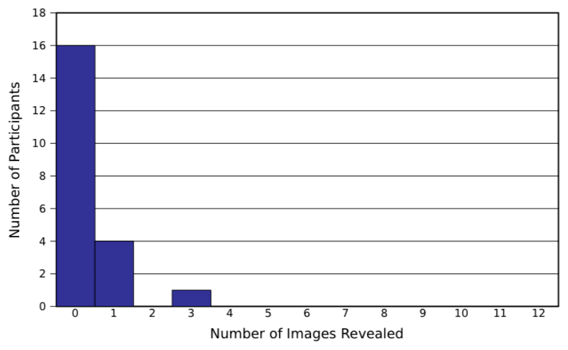
We conducted our user-study over the course of three weeks. We recruited 45 participants from a Craigslist posting. To ensure that subjects had sufficient exposure to their password, we only considered subjects that successfully logged in at least 4 times and at least once in the two days prior to receiving the attack email. 23 subjects met this criteria. This retention rate is comparable to other graphical password user studies[21]. We also eliminated from the analysis two subjects who thought our attack email was a real phishing attack. Participants ranged in age from 18 to 39 years and were varied in race (including Asian, African-American, Caucasian and Hispanic) and profession (including students, actors, IT professionals and HR representatives).

We evaluate both the explicit and implicit password information revealed by participants. Participants explicitly revealed part of their password if they found a password image and clicked on it. They revealed information implicitly by searching for images on the phishing site, even if their search was unsuccessful. For example, if a user spends a long time looking through pictures of dogs on the phishing site, then the phisher can infer that one of the user's password images is of a dog.

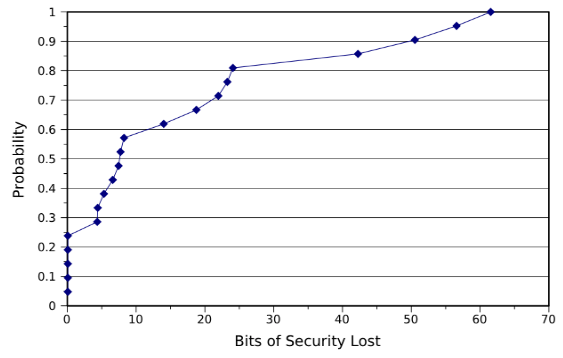
Explicitly-revealed information about σ. Figure 4a shows the number of σ images our participants were able to find and click on. On average, participants clicked on 0.3 images of their σ. No participant clicked on an image that was in their σ but not in their ρ. Furthermore, as Figure 4a shows, 76% participants did not click on any of their σ images, and the others were only able to find at most three of their σ images, implying that PhorceField offers strong protection for almost everyone. During the attack, several participants clicked on random images out of frustration, but doing so provides no useful information to a phisher.

Implicitly-revealed information about σ. Even if users fail to find an image in their σ, they may still reveal information about that image through their search activities. For example, if a victim searches for the term "flowers" during a phishing attack, then the phisher can reduce the search space for one image from 188218 to 3117 images. Furthermore, if that user looks at 10 pages of results without clicking on any of them, then the phisher can conclude that the user's image was not on those pages. If the user performs a second search on the term "plant", the phisher can intersect the two results sets to further narrow the candidate set.

Users may also click on images that are not in their σ but are visually or semantically similar to images that are in their σ. We used the tags on the images users clicked in order to approximate this information, since the tags assigned by Flickr users cover both visual and semantic aspects of the images. Therefore, we took the tags on each clicked image and added them as search queries during the analysis described below. However, we discovered that many of the tags conveyed contex-



(a) The distribution of the number of images in σ that participants clicked on. No user clicked on an image in $\sigma \setminus \rho$.



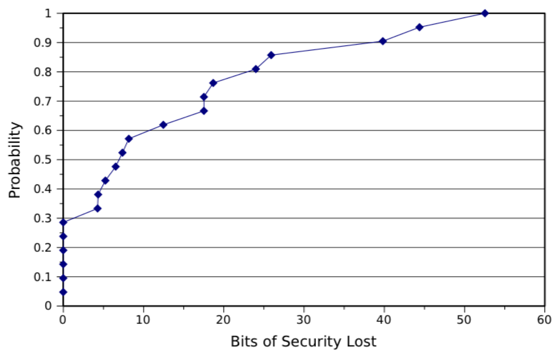
(b) The cumulative distribution function of σ entropy loss for participants in our study.

Figure 4: Explicit and implicit information revealed about σ.

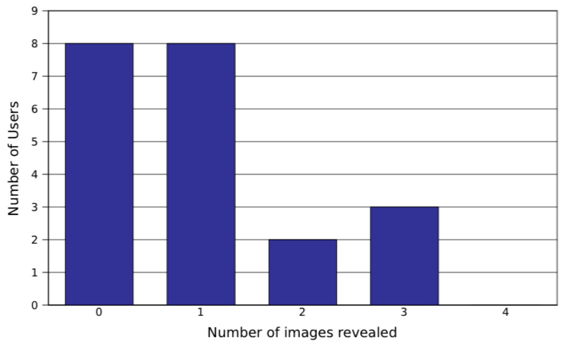
tual information, such as the type of camera used to create the photograph, that would not be apparent to a phishing victim and therefore would not contribute useful information to a phisher. A real phisher could clean the image tags to avoid this problem, but we took a shortcut: For each image a participant clicked on, we computed the set of tags on that image that also occur on an image in the participant's σ and treat those tags as additional search queries performed by that participant. In reality, a phisher would not know which tags occur in the user's σ, so this simplification grossly overestimates the information gained by a phisher.

Given the set of search queries performed by a participant, we upper bound the information gained by the phisher as follows. For each search query, q, let S_q be the set of images in the search result set, and let $U_q \subseteq S_q$ be the set of images in S_q that the user never looked at. For search queries derived from tags on clicked images, $U_q = S_q$. For each image $\iota \in \sigma$, let

$$C_\iota = \begin{cases} \{\iota\} & \text{if user clicked on } \iota \\ \Sigma & \text{if } \forall q. \iota \notin S_q \\ \bigcap_{q:\iota \in U_q} U_q \cap \bigcap_{q:\iota \in S_q \setminus U_q} S_q & \text{otherwise} \end{cases}$$



(a) The cumulative distribution function of entropy loss for ρ from participants in our study.



(b) The distribution of the number of password images implicitly revealed by participants in our study.

Figure 5: The implicitly-revealed information about ρ. In (b), we assume the attacker steals σ after interacting with the victim. Even in that case, he is not able to learn the victim's entire password.

In other words, for each image, we take all the search results that contained that image and intersect them. We also remove any images the user looked at in that search set, unless the user overlooked ι, in which case we take the whole result set. We then used the C_ι values to bound the entropy loss for each user's σ (details of this computation will be in a companion tech report). Note this estimate is conservative because it assumes the phisher knows which search queries contained each image and whether the user overlooked an image in each results set.

Figure 4b shows the cumulative distribution function of entropy loss experienced by participants in our study. No participant revealed more than 62 bits of information about their σ, giving them a residual security of at least 119 bits. Furthermore, more than 80% of participants lost less than 25 bits of σ security.

Implicitly-revealed information about ρ. To estimate the amount of information a phisher can gain about a user's password, we compute C_ι as above for each $\iota \in \rho$, and compute an upper bound on the entropy loss as above. Note that this analysis is conservative for the same reasons as before, plus it conservatively assumes the phisher can infer which candidate sets, C_ι correspond to images in ρ, and the position of each image in ρ. Figure 5a shows the CDF of bits lost on ρ. On average, participants revealed only 13.8 bits of information about their password. No participant revealed more than 52.6 bits of information (out of 70 bits) about her password, and 85% of our participants revealed less than 30 bits of information. PhorceField provided strong protection for all our participants' passwords.

Implicitly-revealed information about $\rho|\sigma$. Finally, to demonstrate the resiliency of PhorceField, we assume the attacker gains access to σ after conducting the phishing attack. Although a real attacker could attempt to conduct a second phishing attack in this case, the analysis below is intended to show that phishing attacks without σ reveal very little information about user passwords. In this case, we consider an image $\iota \in \rho$ to be completely revealed if the attacker gains any information about it during the phishing attack, i.e. if $C_\iota \neq \Sigma$. Note this assumes the attacker can tell which images are in ρ versus $\sigma \setminus \rho$. Figure 5b shows the distribution of the number of password images revealed in this scenario. Over 75% of the participants revealed fewer than 2 of their password images, and no one revealed all 4 of their password images. If we assume the phisher knows the location of each revealed image within the user's password, then our participants revealed, on average, 3.52 bits of information about $\rho|\sigma$.

Other observations. As mentioned above, we asked participants at the end of the study whether they suspected our email was part of a phishing attack, and we removed two subjects who refused to visit our password recovery page due to phishing suspicion. We also tested users to confirm that they still remembered their passwords – only one subject failed this test.

7. DISCUSSION

The results of our user study imply that PhorceField will prevent most password phishing attacks from succeeding. 76% of our participants failed to find even a single image in their password, and no participant found his entire password. Even assuming the phisher later gained access to σ, no participant entirely compromised his password. This compares quite favorably to passive phishing defenses such as SiteKey[4]. Previous studies have shown that 92% of users will reveal their entire text password to a phisher, even if their SiteKey is missing. In our study, 0% of participants revealed their entire password.

Our results suggest reasonable security parameters for a real-world PhorceField deployment. Our choice of $|\Sigma| \approx 2 \times 10^5$ was sufficiently large to make searching for images difficult. Likewise, selecting about a thousand images for each concrete noun made examining search results a tedious task, improving phishing-resistance. Future deployments can increase security without harming usability by choosing a larger Σ. For example, there are over 100 million Creative Commons licensed photos on Flickr[30], so it would be straightforward to construct a Σ with over 100 million images representing over 10000 concrete nouns. Given that some participants revealed 3 of their password images, $|\rho|$ should be at least 6 and possibly higher. Passwords should not allow repeated images, since this forces users to find the maximum possible number of images during a phishing attack.

Most users gave up after trying to find 1 or 2 images, so their search queries only revealed information about 1.33 images in their σ on average. Phishers could design attacks to avoid this bottleneck, but the success rate may or may not improve. For example, a phisher could present victims with 12 text boxes and ask the victim to describe each of the images in their σ. By forcing victims to describe their entire σ instead of allowing them to focus on one image at a time, the attacker may gain information about more images in σ. However, the information gained about each image would probably be less specific, e.g. the victim might simply enter "bird" instead of performing several searches such as "bird", "flying", and "flock", which together reveal much more information about the given image. We could render this attack ineffective by using images, such as random art images, that admit no easy description.

8. RELATED WORK

We discussed related anti-phishing technologies in Section 2, so we focus only on related graphical password research here.

Suo, et al., provide an extensive review of graphical password systems and group them into three categories[29]. Cognometric graphical password systems are the most prevalent. The Déjà Vu system[10] asks users to select a subset of images from a collection of random art images. Weinshall, et al., have an interface that shows users 100-200 sets of image where the user password consists of selecting a predetermined image from each of the sets[32]. Systems like Passfaces take advantage of users' innate ability to recall faces to improve recall[8]. Some systems use thumbnails generated from a single image rather than utilizing image sets[16]. Shoulder resistant schemes for graphical password systems have been developed as well. One such interface asks users to click within a convex hull formed by the objects consisting of their passwords[28]. Another shoulder-surfing resistant strategy includes using an eye tracker to determine the sequence of images the user looked upon[20]. Drawmetric graphical password systems require users to reproduce a shared secret to authenticate. One such technique has the user draw a secret on a 2-D grid and reproduce it for authentication. Other similar techniques include presenting a signature provided by either a stylus[14] or mouse[17]. Locimetric graphical passwords have users repeat a sequence of actions. These systems have users click on a series of interesting and meaningful points in an image in a predetermined sequence[24, 33]. Builders of these systems argue that a large password space can be constructed from a single image since images can contain hundreds of memorable points.

9. CONCLUSION

In this paper, we presented PhorceField, a password ceremony designed to depend on human laziness rather than vigilance. PhorceField uses a client-side secret and graphical password scheme to make it effectively impossible for a user to provide her password to a phisher. As long as the phisher does not know the client-side secret, he can only present the user with a non-standard and difficult-to-use password interface. We also designed PhorceField to exploit weaknesses of human memory to make phishing attacks even less likely to succeed.

We conducted a user study to verify that users will be unable to comply with a phisher's requests. No participant in our study successfully entered his entire password into our phishing web page, and the vast majority of participants revealed less than half of their password.

10. REFERENCES

[1] Saeed Abu-Nimeh, Dario Nappa, Xinlei Wang, and Suku Nair. A comparison of machine learning techniques for phishing detection. In *eCrime '07: Proceedings of the anti-phishing working groups 2nd annual eCrime researchers summit*, pages 60–69, New York, NY, USA, 2007. ACM.

[2] M. Aburrous, M.A. Hossain, F. Thabatah, and K. Dahal. Intelligent phishing website detection system using fuzzy techniques. In *Information and Communication Technologies: From Theory to Applications, 2008. ICTTA 2008. 3rd International Conference on*, pages 1–6, April 2008.

[3] M. C. Anderson and J. H. Neely. *Memory. Handbook of Perception and Cognition*, chapter Interference and inhibition in memory retrieval. Academic Press, 2nd edition, 237–313 1996.

[4] Bank of America. SiteKey: Online Banking Security. `http://www.bankofamerica.com/privacy/sitekey/`.

[5] Mihir Bellare, David Pointcheval, and Phillip Rogaway. *Authenticated Key Exchange Secure against Dictionary Attacks*, pages 139–155. Springer, 2000.

[6] Sonia Chiasson, P. C. van Oorschot, and Robert Biddle. A usability study and critique of two password managers. In *USENIX Security '06: Proceedings of the 15th conference on USENIX Security Symposium*, Berkeley, CA, USA, 2006. USENIX Association.

[7] Neil Chou, Robert Ledesma, Yuka Teraguchi, Dan Boneh, and John C. Mitchell. Client-side defense against web-based identity theft. In *NDSS '04: Proceedings of the 11th Annual Network and Distributed System Security Symposium*, February 2004.

[8] Real User Corporation. The Science Behind Passfaces. Technical report, Real User Corporation, June 2004.

[9] Kenneth A. Deffenbacher, Thomas H. Carr, and John R. Leu. Memory for words, pictures, and faces: Retroactive interference, forgetting, and reminiscence. *Journal of Experimental Psychology: Human Learning and Memory*, 7(4):299–305, 1981.

[10] Rachna Dhamija and Adrian Perrig. Déjà vu: a user study using images for authentication. In *SSYM'00: Proceedings of the 9th conference on USENIX Security Symposium*, pages 4–4, Berkeley, CA, USA, 2000. USENIX Association.

[11] Rachna Dhamija and J. D. Tygar. The battle against phishing: Dynamic security skins. In *SOUPS '05: Proceedings of the 2005 symposium on Usable privacy and security*, pages 77–88, New York, NY, USA, 2005. ACM.

[12] John Eng. Sample size estimation: how many individuals should be studied? *Radiology*,

227(3):309–313, 2003.

[13] GeoTrust Inc. `http://www.geotrust.com/comcasttoolbar/`.

[14] Joseph Goldberg, Jennifer Hagman, and Vibha Sazawal. Doodling our way to better authentication. In *CHI '02: CHI '02 extended abstracts on Human factors in computing systems*, pages 868–869, New York, NY, USA, 2002. ACM.

[15] Rick Hodgin. Phishing cost the u.s. $3.2 billion in 2007. `http://www.tomshardware.com/news/phishing-cost-u-s-3-2-billion-2007,4576.html`, December 2007.

[16] Wayne Jansen. Authenticating mobile device users through image selection, May 2004.

[17] Ian Jermyn, Alain Mayer, Fabian Monrose, Michael K. Reiter, and Aviel D. Rubin. The design and analysis of graphical passwords. In *SSYM'99: Proceedings of the 8th conference on USENIX Security Symposium*, pages 1–1, Berkeley, CA, USA, 1999. USENIX Association.

[18] Chris Karlof, J.D. Tygar, and David Wagner. Conditioned-safe Ceremonies and a User Study of an Application to Web Authentication. In *Sixteenth Annual Network and Distributed Systems Security Symposium (NDSS 2009)*, February 2009.

[19] Brian Krebs. Citibank phish spoofs 2-factor authentication. `http://blog.washingtonpost.com/securityfix/2006/07/citibank_phish_spoofs_2factor_1.html`, July 2006.

[20] Manu Kumar, Tal Garfinkel, Dan Boneh, and Terry Winograd. Reducing shoulder-surfing by using gaze-based password entry. In *SOUPS '07: Proceedings of the 3rd symposium on Usable privacy and security*, pages 13–19, New York, NY, USA, 2007. ACM.

[21] Wendy Moncur and Grégory Leplâtre. Pictures at the atm: exploring the usability of multiple graphical passwords. In *CHI '07: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 887–894, New York, NY, USA, 2007. ACM.

[22] Tyler Moore and Richard Clayton. An empirical analysis of the current state of phishing attack and defence. In *In Proceedings of the 2007 Workshop on the Economics of Information Security (WEIS)*, 2007.

[23] Netcraft. Anti-phishing toolbar. `http://toolbar.netcraft.com/`.

[24] L.D. Paulson. Taking a graphical approach to the password. *Computer*, 35(7):19–19, Jul 2002.

[25] Matt Richtel and Verne G. Kopytoff. E-mail fraud hides behind friendly face. *The New York Times*, 2011.

[26] Mehran Sahami, Susan Dumais, David Heckerman, and Eric Horvitz. A bayesian approach to filtering junk e-mail. In *Learning for Text Categorization: Papers from the 1998 Workshop*, Madison, Wisconsin, 1998. AAAI Technical Report WS-98-05.

[27] Stuart E. Schechter, Rachna Dhamija, Andy Ozment, and Ian Fischer. The emperor's new security indicators. In *SP '07: Proceedings of the 2007 IEEE Symposium on Security and Privacy*, pages 51–65, Washington, DC, USA, 2007. IEEE Computer Society.

[28] Leonardo Sobrado and Jean-Camille Birget. Graphical passwords. In *The Rutgers Scholar, An Electronic Bulletin of Undergraduate Research*, volume 4, 2002.

[29] Xiaoyuan Suo, Ying Zhu, and G. Scott. Owen. Graphical passwords: A survey. In *ACSAC '05: Proceedings of the 21st Annual Computer Security Applications Conference*, pages 463–472, Washington, DC, USA, 2005. IEEE Computer Society.

[30] Michelle Thorn. Analysis of 100m cc-licensed images on flickr. `http://creativecommons.org/weblog/entry/13588`, March 2009.

[31] Yue Wang, R. Agrawal, and Baek-Young Choi. Light weight anti-phishing with user whitelisting in a web browser. In *Proceedings of the 2008 IEEE Region 5 Conference*, pages 1–4, April 2008.

[32] Daphna Weinshall and Scott Kirkpatrick. Passwords you'll never forget, but can't recall. In *CHI '04: CHI '04 extended abstracts on Human factors in computing systems*, pages 1399–1402, New York, NY, USA, 2004. ACM.

[33] Susan Wiedenbeck, Jim Waters, Jean-Camille Birget, Alex Brodskiy, and Nasir Memon. Authentication using graphical passwords: Basic results. In *Human-Computer Interaction International (HCII 2005)*, New York, NY, USA, 2005. Springer.

[34] Yahoo! Welcome to flickr. `http://flickr.com/`.

[35] Yue Zhang, Serge Egelman, Lorrie Cranor, and Jason Hong. Phinding phish: Evaluating anti-phishing tools. In *NDSS '07: Proceedings of the 14th Annual Network and Distributed System Security Symposium*, February 2007.

Dynamic Sample Size Detection in Continuous Authentication using Sequential Sampling

Ahmed Awad E. Ahmed
Department of Electerical and Computer Engineering
University of Victoria
Victoria, BC, Canada
aahmed@ece.uvic.ca

Issa Traore
Department of Electerical and Computer Engineering
University of Victoria
Victoria, BC, Canada
itraore@ece.uvic.ca

ABSTRACT
Continuous Authentication (CA) departs from the traditional static authentication scheme by requiring the authentication process to occur multiple times throughout the entire logon session. One of the main objectives of the CA process is to detect session hijacking. An important requirement about designing or operating a CA system is the need to achieve the quickest detection while maintaining rates of missed and false detections to predetermined levels. We introduce in this paper a new approach for detection based on the sequential sampling theory that allows balancing appropriately between detection promptness and accuracy in CA systems. We study and illustrate the proposed approach using an existing mouse dynamics biometrics recognition model and corresponding sample experimental data.

Keywords
Continuous Authentication, Biometrics, Security Monitoring, Sequential Sampling, Mouse Dynamics.

1. INTRODUCTION
User authentication is the process of verifying whether the identity of a user is genuine prior to granting him access to resources or services in a computer system. Traditionally, authentication is performed statically at the point of entry to the system (e.g., login). A successful authentication, however, at the beginning of a session does not provide any remedy against the session being hijacked later by some malicious user. One of the solutions proposed to address this shortcoming is *continuous authentication (CA)*. CA consists of the process of positively verifying the identity of a user in a repeated manner throughout a computing session to ensure that the claimed identity (at login time) remains the same as the monitored one. In this case, the detection delay or length of an individual authentication period can be captured in terms of either the time or the amount of data involved. In this paper, we refer to the detection delay as the *time-to-detect (TTD)*. One of the important questions that will be raised while designing a CA system is when to stop data collection and detect or, in other words, what should be the detection delay? For some systems, the answer to the above questions can be simple if the data arrival rate is known in advance as well as the entropy of the collected data. A model can be tuned to obtain an acceptable level of error for an optimal TTD. For other systems such as mouse dynamics, the rate of data arrival is unknown and the effectiveness of the collected data is unpredictable. One user can take 5 minutes to produce 1000 actions that cover only a small part of the detectable factors, while another user can take double the time to produce a lower number of actions covering all of the detectable factors. For such systems, there is a risk of achieving poor accuracy if the aim is to minimize the TTD. In this paper, we present a new approach that can be used to solve this dilemma. The approach is based on the sequential sampling technique [12] and provides a systematic procedure to determine the time needed to make the decision. The technique assures a predetermined level of error and has the flexibility to tune it according to the planned security needs. Although, we illustrate and validate the approach using mouse dynamics biometric modality, we believe that the approach could readily be extended to some other behavioral biometric modalities such as keystroke dynamics. The remainder of the paper is structured as follows. In Section 2, we summarize and discuss related work. In Section 3, we provide some background on sequential sampling technique and summarize an existing mouse dynamics biometric model and dataset used in our study. In Section 4, we introduce our technique for dynamic sample size detection based on sequential sampling. In Section 5, we present the experimental evaluation of the proposed technique. Finally, in Section 6, we make some concluding remarks and discuss future work.

2. RELATED WORKS
In the last decade, continuous authentication has been dealt with *indirectly* in various areas including user command sequence monitoring [7], free-text detection of keystrokes dynamics biometrics [3,4], and mouse dynamics biometrics recognition [1,9]. The basic assumption underlying such systems is that illegitimate activity can be discriminated from normal user behavior when based on human-computer interactions. For instance, in the case of command string monitoring the recognition task consists of classifying sequences, or blocks, of user commands as pertaining to "self" (i.e. legitimate user), or "nonself" (i.e. unauthorized user) [7].

CA has also been directly the focus of several proposals in the recent literature. Ikehara and Crosby proposed a continuous identity authentication system (CIAS) that uses multimodal sensors to monitor and protect access to critical computer systems [5]. The proposed system is intended for the protection of critical systems with a limited number of users (e.g., 1 to 3 users). The multimodal sensors consist of a set of passive sensors built into a computer mouse measuring the pressures applied to the mouse during clicking when performing a task of varying difficulty. According to Ikehara and Crosby, a pilot study involving six participants yielded 95% accuracy after two clicks when using discriminant analysis.

Sim *et al*. proposed a multimodal CA system that combines a digital camera-based facial recognition scheme with a mouse-based fingerprint recognition scheme [10]. The system involves a feedback mechanism into the operating system that automatically locks up the computer by delaying processes or suspending them entirely. The mouse sensor is a modified computer mouse that incorporates an optical fingerprint scanner at the place where the user would position their thumb when using the mouse. The authors also proposed a holistic fusion technique using Hidden Markov Model (HMM) that integrates the face and fingerprint across modalities and time for CA. It was claimed that the proposed fusion scheme captures more suitably the characteristics of CA systems than traditional ones. Azzini and Marrara proposed a multimodal biometric CA system that combines face and fingerprint recognitions using a fuzzy controller [2]. After the initial authentication, the user identity is continuously checked on the sole basis of the facial recognition, until the computed score falls below a certain threshold, at which point fingerprint recognition is triggered.

Although the need for balancing the goals of detection promptness and detection accuracy has largely been recognized in the CA literature (including the above proposals), we have not come across any work that directly addresses this problem from a CA perspective. This issue, however, has already been tackled in the intrusion detection field as a sequential decision problem from different angles. In [6], Jung *et al*. introduced an approach for accurate portscans detection with minimal delay based on sequential sampling theory. In the proposed approach, accesses to local IP addresses are modeled as a random walk on one of two stochastic processes corresponding to traffic originating either from benign hosts or scanners, respectively. In [11], Tartakovsky *et al*. used sequential change-point detection theory for timely and accurate detection of abrupt changes in the network traffic, which potentially may be linked to attacks such as denial-of-service or portscans. While sequential hypothesis testing allows differentiating between normal and malicious behaviors, sequential change-point detection determines the specific point where normal behavior starts becoming malicious. Although both types of problems are relevant to continuous authentication, our primary focus in this paper is on the former. In this regard our approach is closely related to the work of Jung *et al*., although we target different types of attacks and use different types of data.

3. BACKGROUND

3.1 On Mouse Dynamics Biometric Analysis

Mouse dynamics correspond to the actions generated by the mouse input device for a specific user while interacting with a graphical user interface. Many characteristics of the mouse movement can be used for biometrics analysis. In [1], the following characteristics for each mouse movement are considered: Type of action, Traveled distance (in pixels), Elapsed Time (in seconds), and Movement Direction. Three types of actions are considered, namely: point-and-click (PC), drag-and-drop (DD), and regular mouse movement (MM). The movement direction is determined based on the angle of the movement. Eight directions, numbered from 1 to 8, are considered. Each direction covers an area of 45 degrees moving clockwise from the forward direction. In [1], the following seven biometric factors were extracted from the raw mouse data and used for biometric identification:

- The Movement Speed compared to Traveled Distance (denoted MSD) factor: computed by approximating the raw mouse data to a curve using Neural Networks.
- The Average Movement Speed per Movement Direction (denoted MDA) histogram.
- The Movement Direction Histogram (MDH): corresponds to the distribution of actions in each of the eight directions considered.
- The Average Movement Speed per Type of Action (denoted ATA).
- The Action Type Histogram (ATH): corresponds to the distribution of the performed actions over the different types of actions considered.
- The Traveled Distance Histogram (TDH): represents the distribution of the number of actions performed by the user within different distance ranges.
- The Movement elapsed Time Histogram (denoted MTH): represents the distribution of the number of actions performed by the user within different time ranges.

To build a biometric profile for the user, 39 features, each corresponding to a point judiciously chosen from the above curve and histograms were used. To compare some monitored behavior against a reference profile, a feed-forward neural network consisting of three layers was used.

The neural network takes as input the 39-dimensional feature vector and computes as output a percentage referred to as the confidence ratio (CR), which represents the degree of similarity of the compared behaviors. During enrolment, the above same neural network architecture is trained for each user, and a profile is created for each user based on the weights of the neural network. During the detection mode, the weights of the neural network are restored from the profile database and the newly collected mouse samples are applied to the network. The output is then compared to a preset threshold in order to decide on the similarity or dissimilarity of the compared behaviors. The above scheme was evaluated through free and controlled experiments involving 22 and 7 users, respectively. In the free experiment, the participants were given an individual choice of operating conditions and applications. Consequently, data was collected using a variety of hardware and software systems. The experiment ran for 9 weeks, and allowed the collection of 284 hours of raw mouse data over 998 sessions, with an average of 45 sessions per participant. By analyzing the data, an Equal Error Rate of 2.46% was obtained when setting the CR threshold to 50%. The objective of the controlled experiment was to study the impact of confounding factors involved in the free experiment. Participants were asked to perform the same set of actions using a customized application deployed on the same operating environment (i.e. same machine, mouse and operating system). Data was collected over 63 sessions, 9 sessions per participants, and each session consisting of 100 mouse actions. A FAR of 2.245% and a FRR of 0.898% was achieved, when the threshold was set to CR=50%.

3.2 On Sequential Sampling Technique

To our knowledge, previous CA models were all based on classical sampling where the sample data is predetermined.

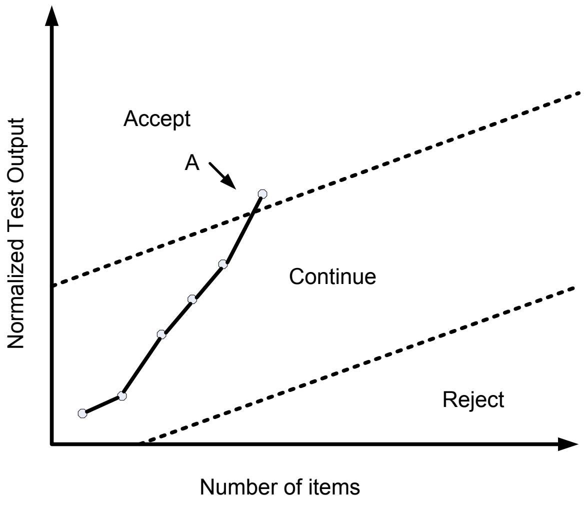


Figure 1. Sequential sampling plan.

In classical statistical techniques, the sample size is set according to the expected level of features prevalence over the data and the degree of precision required. At the end of the data collection process, a decision will be made to accept or reject the null hypothesis. The sequential sampling technique, originally proposed by Wald, combines (simultaneously) data collection and data analysis into a single process (sampling plan) [12]. In this approach, the sample size is not fixed — the size increases over time until a decision is made. Hypothesis testing can occur every time data is available (item by item unit-step sequential sampling) or over incremental batches of data (group sequential sampling). Applying the sequential sampling technique requires building a sampling plan consisting of three areas (*Accept*, *Continue*, and *Reject*). A test will be conducted on the data collected up to the current sampling period and, according to the result of this test, the decision will be made. Figure 1 shows the sampling plan. The plan is partitioned into three regions. The x-axis represents the number of items included in the evaluation, and the y-axis represents the output of the test performed on the sampled data. Each time a sample is tested, a point will be placed on the plan and the status will be evaluated according to the location of the point on the plan. The sampling process will continue (adding more items to the sample and testing again) if the point falls in the *Continue* region, and will stop if it falls in the *Accept* or *Reject* regions.

The sampling plan is built according to the following parameters:

α = acceptable type I error or false rejection rate (FRR).

β = acceptable type II error of false acceptance rate (FAR).

p_1 = lower threshold limit (as proportion).

p_2 = higher threshold limit (as proportion).

The two lines representing the borders between the *Accept*, *Continue*, and *Reject* areas can be described as:

$$N_{CR} = -h_1 + sn \text{ (Rejection line)}$$

$$N_{CR} = h_2 + sn \text{ (Acceptance line)}$$

where n is the test number (x-axis) and N_{CR} is the normalized value of the confidence ratio (CR) calculated for test number n;

$$N_{CR} = CR_n \times n/100 \ .$$

Parameters h_1, h_2 and s can be computed as follows:

$$h_1 = \left(\ln \frac{1-\alpha}{\beta}\right) \Big/ \left(\ln \frac{p_2(1-p_1)}{p_1(1-p_2)}\right);$$

$$h_2 = \left(\ln \frac{1-\beta}{\alpha}\right) \Big/ \left(\ln \frac{p_2(1-p_1)}{p_1(1-p_2)}\right);$$

$$s = \left(\ln \frac{1-p_1}{1-p_2}\right) \Big/ \left(\ln \frac{p_2(1-p_1)}{p_1(1-p_2)}\right).$$

On the sampling plan, the variables h_1 and h_2 affect the distance between the two border lines while s represents their slope.

4. DYNAMIC SAMPLE SIZE DETERMINATION

We present in this section our approach for dynamic sample size determination using sequential sampling. Before describing the technique, let us try to evaluate how the system accuracy is affected with respect to the time needed for the data to arrive. Figure 2 shows how CR changes as the number of actions included in the evaluation increases. Each point on the curve represents the CR computed by processing $n \times N$ mouse actions (from the dataset collected in [1]), where n is the test number and N is the number of actions in a sampling period. Exactly 50 tests were conducted; therefore the last point on the curve represents the CR resulting from processing 2500 actions ($n = 50$ tests, $N = 50$ actions).

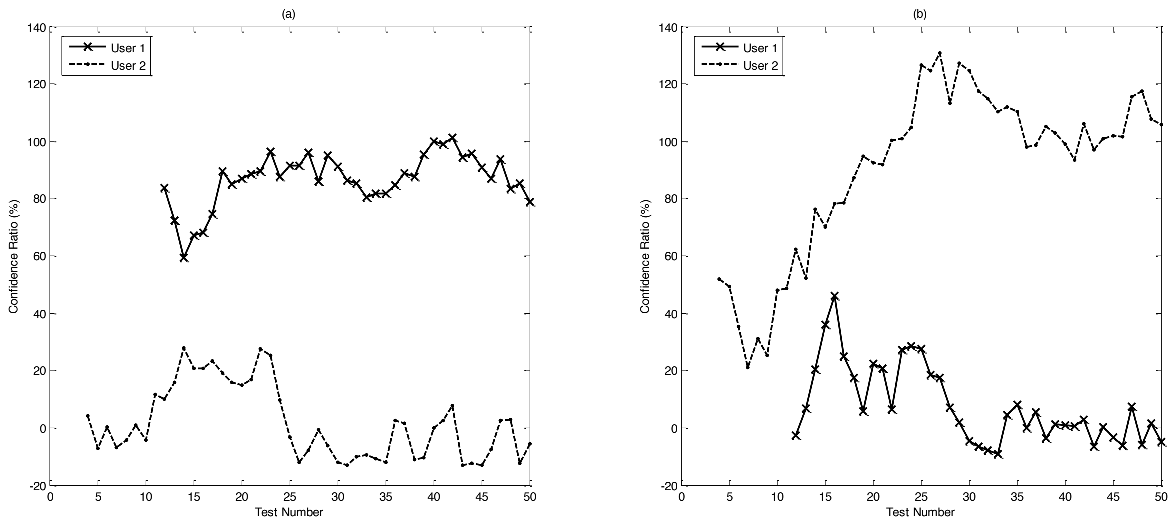


Figure 2. Evolution of CR as the number of actions increases; earlier test points are missing because the sample size is too small to compute the CR. a. Confidence ratio resulted from testing user 1's and 2's data against User 1's reference signature. b. Confidence ratio resulted from testing User 1's and 2's data against User 2's reference signature.

Figure 2-a shows two curves for two different users — User 1 and User 2 — both compared to User 1's reference signature. Figure 2-b shows the results when the data is compared to User 2's reference signature. Note that for both users the earlier tests failed to compute the CR because of the low number of data available; this explains why corresponding test points are missing. Considering a simple detection algorithm that will base its result on whether the resulting CR is below or above a threshold limit of say CR= 50%, we notice that three out of the four curves give a correct result near the beginning of testing when only a low number of actions are included. We also notice an increasing pattern in the CR as the number of action increases when the session belongs to the reference user, and a decreasing pattern when the current session is not for the legitimate user. As for the fourth curve representing User 2's results in Figure 2-b, there is a probability for false rejection if the decision is made based on the tests where $n < 10$ as the resulting CR will be lower than 50%. Figure 3 illustrates the use of the sequential sampling technique on the two users' data presented in Figure 2.

The CR calculated for each test will be normalized and plotted on the plan. As the number of actions increases, the algorithm will give more weight to the calculated CR and at the same time will increase the limit required to be passed to accept and decrease the one needed to reject. The decision is made as soon as the normalized CR value crosses one of the limits (example: point A on Figure 1); at that time, the sampling process will stop and a new sampling process will start for the data to come after. Note that the technique avoided the problem mentioned above regarding the evaluation of user 2's data on Figure 2-b before $n = 10$, which can lead to a false rejection. As shown in Figure 3-b, the technique waited until $n = 23$ to make the correct decision.

Table 1 demonstrates how the decisions are made for the two sampling processes presented in Figure 3-a. User 1's session is accepted when the normalized CR reaches 25.911, passing the 24.949 limit set by the algorithm to stop sampling at step number 27. The algorithm was able to classify and reject User 2's session at step number 25 when his normalized CR value reached -0.83718, falling in the reject area.

Table 1. Data collected through the sampling process (depicted by Figure 3-a) and the decisions taken in reference to the rejection and acceptance limits. Earlier test points are missing because the sample size is too small to compute the CR.

Test Number	Number of Actions	Reject Limit	Accept Limit	User 1's Normalized CR	User 2's Normalized CR	Decision
1	50	-10.949	11.949	X	X	X
2	100	-10.449	12.449	X	X	X
3	150	-9.9494	12.949	X	X	X
4	200	-9.4494	13.449	X	0.16447	X
5	250	-8.9494	13.949	X	-0.36659	X
6	300	-8.4494	14.449	X	0.009224	X
7	350	-7.9494	14.949	X	-0.48361	X
8	400	-7.4494	15.449	X	-0.35372	X
9	450	-6.9494	15.949	X	0.080125	X
10	500	-6.4494	16.449	X	-0.43563	X
11	550	-5.9494	16.949	X	1.2663	X
12	600	-5.4494	17.449	10.043	1.203	X
13	650	-4.9494	17.949	9.4186	2.0601	X
14	700	-4.4494	18.449	8.3071	3.9064	X
15	750	-3.9494	18.949	10.059	3.109	X
16	800	-3.4494	19.449	10.898	3.302	X
17	850	-2.9494	19.949	12.66	3.9447	X
18	900	-2.4494	20.449	16.109	3.4102	X
19	950	-1.9494	20.949	16.14	3.0154	X
20	1000	-1.4494	21.449	17.371	2.9798	X
21	1150	-0.9494	21.949	18.611	3.509	X
22	1200	-0.4494	22.449	19.664	6.0904	X
23	1250	0.05059	22.949	22.137	5.781	X
24	1300	0.5506	23.449	21.046	2.3659	X
25	1350	1.0506	23.949	22.847	-0.83718	Reject User 2
26	1400	1.5506	24.449	23.806	X	X
27	1450	2.0506	24.949	25.911	X	Accept User 1

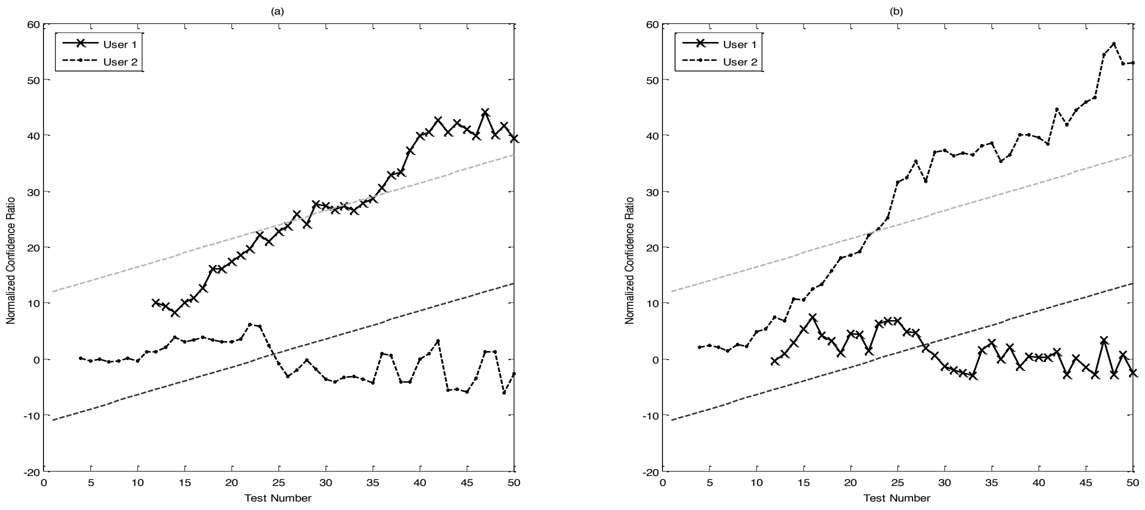


Figure 3. Sequential sampling plans showing: (a) test results for User 1 and User 2 when compared to User 1's reference signature; (b.) test results for User 1 and User 2 when compared to User 2's reference signature. Earlier test points are missing because the sample size is too small to compute the CR.

The sampling plan is established by choosing values for the parameters that properly describe the classification criteria and the acceptable level of error. For example, in our first test we chose $N = 50$, $\alpha = 0.01$, $\beta = 0.01$, $P_1 = 0.45$, and $P_2 = 0.55$. This means that the algorithm will accept the hypothesis that the user session is legitimate if the calculated CR passes the 55% threshold limit, and will reject the hypothesis indicating an intrusion if the CR value is lower than 45%. If the CR falls in the range between the two limits, a decision cannot be made and the sampling will continue until CR reaches one of the two limits. In addition, according to those parameters, there is a probability of 1% that the algorithm will misclassify a session (due to the fact that $\alpha = \beta = 0.01$).

5. EXPERIMENTAL EVALUATION

The sequential sampling technique described above can help in decreasing the time needed to make a decision about the provided session data; a factor that can be very critical in CA applications. The technique also can help in increasing the accuracy since a decision is made incrementally and based on a number of repetitive evaluations, not only a single one. Prior to the application of this technique, it is very important to select the proper sequential sampling parameters to ensure the achievement of the potential goal. The selection of these parameters can be made theoretically by setting the aimed values for the accuracy (FAR, FRR); however, it is not possible to set an expected value for the TTD. Because the sequential sampling process involves repetitive processing of the data, another important factor to be considered is the CPU usage. In order to study the effect of the sequential sampling parameters on the factors mentioned above, we conducted an experimental evaluation using the mouse dynamics data collected in [1]. We describe in this section the experimental method, and present and analyze the results.

5.1 Methods

Our evaluation of the processing requirement of the tests indicated a very high demand for processing power. Around 40 days are needed to finish processing the data for one test on an Intel dual Xeon machine with 2 GB of RAM. In order to avoid falling into this problem, we considered a small subset of the data where we randomly selected 5 users and included only 20,000 actions for each of them. Similar to [1] a one-hold-out cross-validation test was performed on the data, where 2000 user actions per user were used to calculate the users' signatures in each round. The rest of the data were used to simulate legitimate/attack sessions. In each round, one of the users was considered as an impostor, while the other four users played the role of insiders. A profile was built for an insider by using 2000 actions from their own data for positive training while 2000 actions from each of the three remaining insiders were used for negative training. FAR was computed by comparing impostor data against insiders' profiles, while FRR was computed by comparing each insider's profile against their own data. The main difference between this evaluation and the one performed in [1] is that the session size is not fixed. In this evaluation the decision is not made based on a specific number of actions in a session. Instead, the data is sequentially examined until the normalized CR reaches one of the decision limits. By then, a new session will start. With this limitation, it takes an average of 8 hours to process the data for each test with specific values for the sampling parameters and calculate the resulting factors.

5.2 Results

Figure 4 shows the results obtained from the tests for different values for the parameters N, α, β, P_1, and P_2. We started our evaluation by fixing the parameters α and β to 0.01 in order to study the effect of the other parameters N, P_1, and P_2. We considered five different pairs of values for (P_1, P_2), and for each pair we considered three different values for the sample size N = *25, 50, 100*. So, in total, we conducted 15 different tests. Each point on the graphs represents a test (15 tests in total). For each test, we calculated the FAR, FRR, Mean TTD (denoted MTTD), expressed as the number of actions needed to make the final decision and the average CPU usage needed to process the data in each of the sampling tests. The aim was to lower these values as much as possible or to balance between them according to the prospective application.

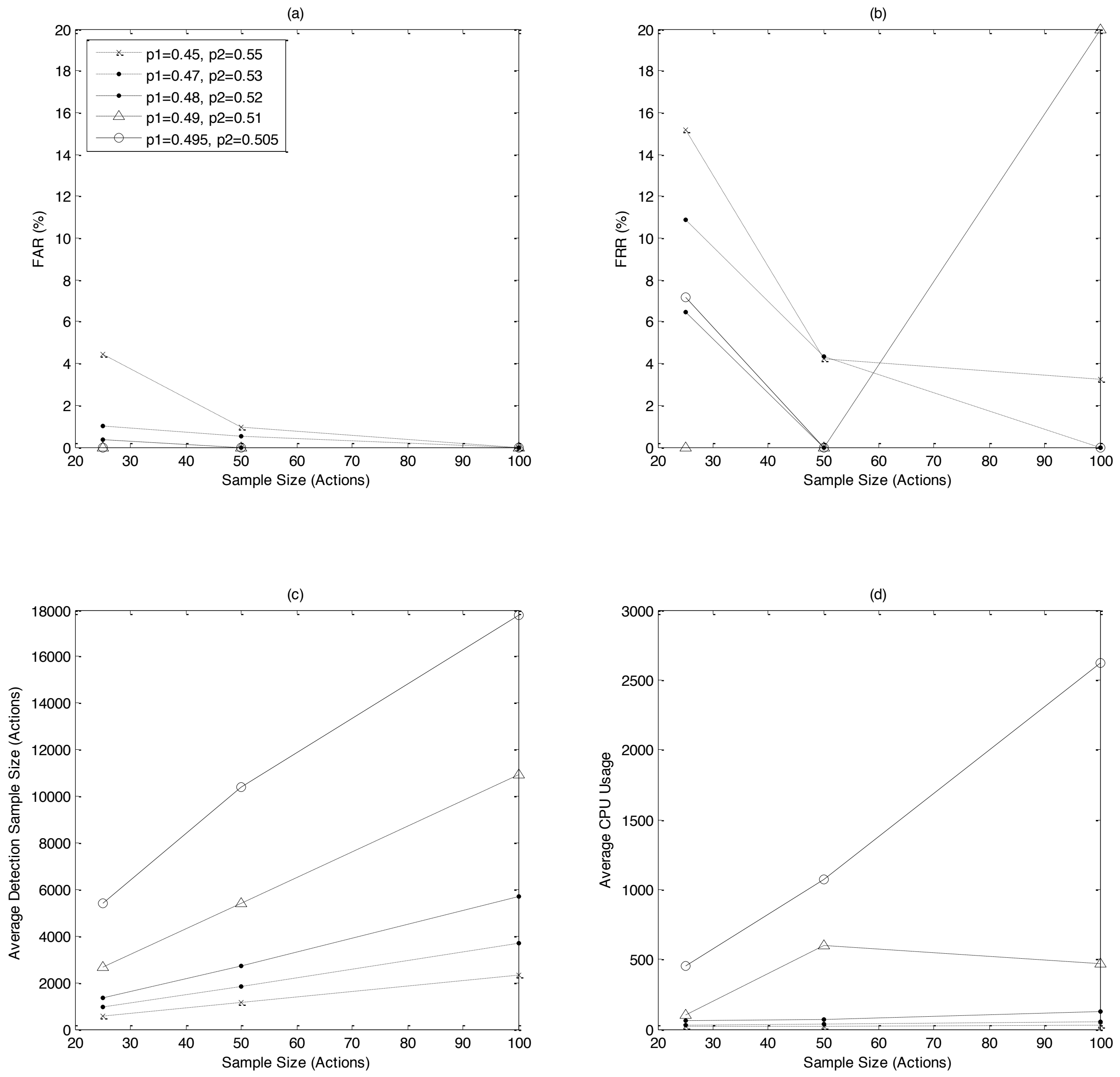


Figure 4. Tuning sequential sampling parameters by studying their effect on: (a) FAR, (b) FRR, (c) MTTD, and (d) CPU usage.

Looking at Figure 4-a, we notice a decreasing pattern for the FAR as the number of actions in the sampling window (i.e. parameter N) increases. We also note that the range that FAR swings in also decreases as the parameter N increases. The same comments apply to FRR (Figure 4-b), except for the test where ($P_1 = 0.49$, $P_2 = 0.51$) where FRR jumps from 0 to 20% as N increases from 50 to 100 actions. For all tests, the values of FRR are always greater than or equal to the corresponding values of FAR.

From Figure 4-c, we note that the MTTD (mean of the number of actions in all detection sessions) increases as the parameter N increases and as the difference between P_1 and P_2 decreases. Only 5 tests out of the 15 resulted in a MTTD lower than 2000 actions. The CPU usage (Figure 4-d) also increases as N increases, and becomes very high only for N greater than 25 actions and when the difference between the parameters P_1 and P_2 is 0.2 or lower.

Table 2 shows the resulting values for FAR, FRR, MTTD and CPU usage for the tests shown in Figure 4. The table also shows the minimum and maximum TTD obtained for all of the tests.

Through analysis of this table, and by considering the acceptable ranges for these values, we can select the tests that satisfy these limits. These values can change according to the requirements of the detection system. If the aim is to detect intrusions as soon as possible, then decreasing MTTD or Min TTD will be the goal, and if the aim is to increase the accuracy then lowering FAR and FRR is the goal. Ideally, we might also like to achieve the lowest possible CPU usage.

Table 2. FAR, FRR, MTTD, and CPU usage achieved for different values of the parameters P_1 and P_2 while setting α and β to 0.01 for N = 25, 50, and 100 actions. Highlighted rows are candidates for improvement towards achieving the selected criteria.

Test Number	Sample Size N (Actions)	P_1	P_2	α	β	FAR (%)	FRR (%)	Min TTD (Actions)	Max TTD (Actions)	Mean TTD (Actions)	CPU Usage (Seconds)
1a	25	0.45	0.55	0.01	0.01	0.04437	0.15172	75	1825	550.24	17.912
2a	25	0.47	0.53	0.01	0.01	0.01033	0.1087	275	2375	933.25	25.165
3a	25	0.48	0.52	0.01	0.01	0.00377	0.06451	300	2875	1350.2	57.718
4a	25	0.49	0.51	0.01	0.01	0	0	1175	5150	2657.9	104.28
5a	25	0.495	0.505	0.01	0.01	0	0.07142	2625	8375	5381.7	451.1
6a	50	0.45	0.55	0.01	0.01	0.00967	0.04225	400	2900	1160	21.82
7a	50	0.47	0.53	0.01	0.01	0.00518	0.04347	800	4450	1816.5	39.337
8a	50	0.48	0.52	0.01	0.01	0	0	1250	5500	2705.2	71.172
9a	50	0.49	0.51	0.01	0.01	0	0	3600	10150	5394.3	599.98
10a	50	0.495	0.505	0.01	0.01	0	0	7000	16450	10405	1071.9
11a	100	0.45	0.55	0.01	0.01	0	0.03225	1000	5600	2328.3	29.4
12a	100	0.47	0.53	0.01	0.01	0	0	1300	7900	3709.2	50.509
13a	100	0.48	0.52	0.01	0.01	0	0	3700	9300	5682.4	125.32
14a	100	0.49	0.51	0.01	0.01	0	0.2	8200	18600	10950	466.78
15a	100	0.495	0.505	0.01	0.01	0	0	15100	19600	17789	2622.4

For instance, if the main requirement for the system is to meet the European Standard for Access Control, which requires FAR less than 0.001% and FRR less than 1% [8], the parameters for test 11a (from Table 2) will be the best pick. In addition to satisfying the above requirement, this particular test achieves the lowest MTTD (2328.3 actions) and CPU usage (29.4 seconds) among all other candidates. In addition to the above criteria, if we add another requirement — setting the MTTD to be less than 2000 actions — we notice that none of the tests shown in Table 2 will satisfy such criteria.

The other parameters α and β, which were not considered in the previous tests, can be used to enhance the results obtained by either reducing the TTD or reducing FAR, FRR, or both. A reduction in the TTD will be at the expense of FAR and FRR. Furthermore, each of these parameters has its own effect on FAR and FRR. Lowering α should help in lowering FAR while lowering β should help in lowering FRR. TTD tends to decrease as α and β increase.

Table 3 shows the results of the tuning process. More tests are performed to illustrate the studied effect. Several (test) candidates, which were close to the above criteria as highlighted in Table 2 (e.g. test 3a), are the targets of this new round of tuning. The original tests copied from Table 2 are highlighted in Table 3 and are followed and preceded by the new tests. In the table, trials to decrease FAR and FRR are done by lowering the values of α and β to 0.005 and then to 0.001, while other trials to decrease the TTD are done by increasing α and β to 0.015. The trials in Table 3 succeeded in satisfying the previous criteria in two of the performed tests (Tests 3b and 6c). Considering the fact that the CPU usage for test 6c is 58% lower than the CPU usage for the other test (3b), our best pick for the mentioned criteria will be the parameters used for test 6c ($N = 50$, $\alpha = 0.001$, $\beta = 0.001$, $P_1 = 0.45$, and $P_2 = 0.55$).

Table 3. Tuning the parameters α and β aiming to minimize FAR, FRR, TTD for the highlighted tests to satisfy the selected criteria.

Test Number	Sample Size N (Actions)	P_1	P_2	α	β	FAR (%)	FRR (%)	Min TTD (Actions)	Max TTD (Actions)	Mean TTD (Actions)	CPU Usage (Seconds)
3a	25	0.48	0.52	0.01	0.01	0.00377	0.06451	300	2875	1350.2	57.718
3b	**25**	**0.48**	**0.52**	**0.005**	**0.005**	**0**	**0.07017**	**300**	**3075**	**1523.3**	**60.950**
3c	25	0.48	0.52	0.001	0.001	0.00574	0.02439	925	4625	1995.5	87.233
4b	25	0.49	0.51	0.015	0.015	0.00714	0.03030	975	4725	2407.1	376.72
4a	25	0.49	0.51	0.01	0.01	0	0	1175	5150	2657.9	104.28
4c	25	0.49	0.51	0.005	0.005	0	0	1275	5475	3030.9	129.12
6a	50	0.45	0.55	0.01	0.01	0.00967	0.04225	400	2900	1160	21.82
6b	50	0.45	0.55	0.005	0.005	0.00361	0.11111	350	2900	1291.2	18.181
6c	**50**	**0.45**	**0.55**	**0.001**	**0.001**	**0**	**0.02127**	**450**	**3350**	**1650.8**	**25.997**
7a	50	0.47	0.53	0.01	0.01	0.00518	0.04347	800	4450	1816.5	39.337
7b	50	0.47	0.53	0.005	0.005	0	0.10811	900	4300	2054.8	35.66
8b	50	0.48	0.52	0.015	0.015	0	0.03333	1050	6500	2479.3	48.177
8a	50	0.48	0.52	0.01	0.01	0	0	1250	5500	2705.2	71.172
11a	100	0.45	0.55	0.01	0.01	0	0.03225	1000	5600	2328.3	29.4
11b	100	0.45	0.55	0.005	0.005	0	0.03333	1300	5600	2640	27.251
11c	100	0.45	0.55	0.001	0.001	0	0	1800	7300	3451.7	41.772
12b	100	0.47	0.53	0.015	0.015	0	0	2100	7000	3519.1	44.652
12a	100	0.47	0.53	0.01	0.01	0	0	1300	7900	3709.2	50.509
12c	100	0.47	0.53	0.005	0.005	0	0.05882	2000	6700	4337.4	63.68

6. CONCLUSIONS

We have presented in this paper a new technique for dynamic sample size determination in continuous authentication based on the sequential sampling approach. Experimental evaluation shows that the proposed technique can be used to minimize the length of the detection period while controlling the detection error rates. Although we have illustrated our approach using mouse dynamics biometric, we believe that it can potentially be adapted for several other types of biometric-based continuous authentication systems. Our goal is to investigate this possibility in our future work by adapting and applying the proposed approach for continuous authentication based on keystroke dynamics biometric.

7. REFERENCES

[1] Ahmed A. A. E. and I. Traore 2007. A New Biometrics Technology based on Mouse Dynamics. *IEEE Transactions on Dependable and Secure Computing*, 4, 3 (July-September 2007), 165-179.

[2] Azzini A. and S. Marrara 2008. Impostor Users Discovery Using a Multimodal Biometric Continuous Authentication Fuzzy System. In *KES 2008*, I. Lovrek et al., Eds. Springer-Verlag Berlin Heidelberg, Part II, LNAI 5178, 371-378.

[3] Dowland, P., Singh, H., and Furnell, S. 2001. A Preliminary Investigation of User Authentication using Continuous Keystroke analysis. In *Proc. 8th IFIP Annual Working Conf. on Information Security Management and Small System Security*, Las Vegas, Nevada.

[4] Gunetti D., and C. Picardi 2005. Keystroke Analysis of Free Text. *ACM Transactions on Information and System Security*, 8, 3 (Aug. 2005), 312-347.

[5] Ikehara C. and M. Crosby 2004. Continuous Identity Authentication using Multimodal Physiological Sensors. In *Biometrics Technology for Human Identification*, A. K. Jain and K. Nalini, Eds. *Proc. of the SPIE*, 5404, 393-400.

[6] Jung J., V. Paxson, A.W. Berger, and H. Balakrishnan 2004. Fast Portscan Detection Using Sequential Hypothesis Testing. In *Proc. IEEE Symposium on Security & Privacy*, 211-225.

[7] Maxion R.A. and T.N. Townsend 2003. Masquerade Detection Augmented with Error Analysis. *IEEE Transactions On Reliability Analysis*, 53, 1 (March 2003), 124-147.

[8] Polemi D. 1997. Biometric Techniques: Review and Evaluation of Biometric Techniques for Identification and Authentication, Including an Appraisal of the Areas where they are most applicable. Technical Report. DOI= ftp://ftp.cordis.lu/pub/infosec/docs/biomet.doc

[9] Pusara, M., Brodley, C.E. 2004. User Re-Authentication via Mouse Movements. In *Proceedings ACM VizSec/DMSEC'04*, October 29, 2004, Washington, DC, USA.

[10] Sim T., R. Jankiraman, and S. Kumar 2007. Continuous Verification Using Multimodal Biometrics. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 29, 4 (April 2007), 687-700.

[11] Tartakovsky A.G., B.L. Rozovsky, R. B. Blazek, and H. Kim 2006. A Novel Approach to Detection of Intrusions in Computer Networks via Adaptive Sequential and Batch-Sequential Change-Point Detection Methods. *IEEE Transactions on Signal Processing*, 54, 9 (Sept. 2006), 3372-3382.

[12] Wald A. 1947. *Sequential Analysis*. J. Wiley & Sons, New York.

Improving Robustness of DNS to Software Vulnerabilities

Ahmed Khurshid, Firat Kiyak, Matthew Caesar
University of Illinois at Urbana-Champaign
{khurshi1, caesar}@illinois.edu, firatkiyak@gmail.com

ABSTRACT

The ability to forward packets on the Internet is highly intertwined with the availability and robustness of the Domain Name System (DNS) infrastructure. Unfortunately, the DNS suffers from a wide variety of problems arising from implementation errors, including vulnerabilities, bogus queries, and proneness to attack. In this work, we present a preliminary design and early prototype implementation of a system that leverages diversified replication to increase tolerance of DNS to implementation errors. Our design leverages software diversity by running multiple redundant copies of software in parallel, and leverages data diversity by sending redundant requests to multiple servers. Using traces of DNS queries, we demonstrate our design can keep up with the loads of a large university's DNS traffic, while improving resilience of DNS.

1. INTRODUCTION

The Domain Name System (DNS) is a hierarchical system for mapping hostnames (e.g., www.illinois.edu) to IP addresses (e.g., 128.174.4.87). The DNS is a ubiquitous and highly crucial part of the Internet's infrastructure. Availability of the Internet's most popular services, such as the World Wide Web and email rely almost completely on DNS in order to provide their functionality. Unfortunately, the DNS suffers from a wide variety of problems, including performance issues [9, 17], high loads [11, 27], proneness to failure [25], and vulnerabilities [7]. Due to the propensity of applications and services that share fate with DNS, these problems can bring significant harm to the Internet's availability.

Much DNS research focuses on dealing with *fail-stop* errors in DNS. Techniques to more efficiently cache results [17], to cooperatively perform lookups [23, 24], to localize and troubleshoot DNS outages [22], have made great strides towards improving DNS availability. However, as fail-stop errors are reduced by these techniques, Byzantine errors become a larger bottleneck in achieving availability. Unlike fail-stop failures, where a system stops when it encounters an error, Byzantine errors include the more arbitrary class of faults where a system can violate protocol. For example, software errors in DNS implementations lead to bogus queries [27] and vulnerabilities, which can be exploited by attackers to gain access to and control DNS servers. These problems are particularly serious for DNS – while the root of the DNS hierarchy is highly *physically* redundant to avoid hardware failures, it is *not* software redundant, and hence multiple servers can be taken down with the same attack. For example, while there are 13 geographically distributed DNS root clusters, each comprised of hundreds of servers, they only run two distinct DNS software implementations: BIND and NSD (see [8] and references therein). While coordinated attacks to DoS these servers are hard, the fact these servers may share vulnerabilities makes these attacks simpler. Not as much work has been done in dealing with such problems in the context of DNS.

In this paper, we revisit the classic idea of using *diverse replication* to improve system availability. These techniques have been used to build a wide variety of robust software, especially in the context of operating systems and runtime environments [10, 12, 13, 18, 19, 28]. Several recent systems have also been proposed to decrease costs of replication, by skipping redundant computations [30], and by eliminating storage of redundant states [16]. However, to the best of our knowledge, such techniques have not been widely investigated in improving resilience of DNS. Applying these techniques in DNS presents new challenges. For example, the DNS relies on distributed operations and hence some way to coordinate responses across the wide area is required. Moreover, the DNS relies on caching and hence a faulty response may remain resident in the system for long periods of time.

In this paper we present *DR-DNS*, a design and early prototype DNS service that leverages diverse replication to mask Byzantine errors. In particular, we design and implement a *DNS hypervisor*, which allows multiple diverse replicas of DNS software to simultaneously execute, with the idea being that if one replica crashes or generates a faulty output, the other replicas will remain available to drive execution. To reduce the need to implement new code, our prototype leverages the several already-existing diverse open-source DNS implementations. Our hypervisor maintains *isolation* across running instances, so software errors do not affect other instances. It uses a simple voting procedure to select the majority result across instances, and includes a cache to offset the use of redundant queries. Voting is performed in the *inbound* direction, to protect end-hosts from errors in local implementations or faulty responses returned by servers higher up in the DNS hierarchy. As our voting mechanism selects the majority result, it is able to protect end-hosts from t faulty replicas if we run $2t + 1$ diverse DNS software replicas side-by-side.

Roadmap: To motivate our approach, we start by surveying common problems in DNS, existing work to address them, as well as performing our own characterization study of errors in open-source DNS software (Section 2). We next present a design that leverages diverse replication to mitigate software errors in DNS (Section 3). We then describe our prototype implementation (Section 4), and characterize its performance by replaying DNS query traces (Section 5). We then consider an extension of our design that leverages existing diversity in the current DNS hierarchy to improve resilience, and measure the ability of this approach in the wide-area Internet (Section 6). Next, we consider Content Distribution Networks and their effects on DR-DNS (Section 6.3). We finally conclude with a brief discussion of related work (Section 7) and future research directions (Section 8).

2. MOTIVATION

In this section, we make several observations that motivate our design. First, we survey the literature to enumerate several kinds of Byzantine faults that have been observed in the DNS infrastructure. Next, we study several alternatives towards achieving diversity across replicas. Finally, we study the costs involved in running diverse replicas.

Errors in DNS software: The highly-redundant and overprovisioned nature of the DNS makes it very resilient to physical failures. However, the DNS suffers from a variety of software errors that introduce correctness issues. For example, Wessels et al. [27] found large numbers of *bogus queries* reaching DNS root servers. In addition, some DNS implementation bugs are *vulnerabilities*, which can be exploited by attackers to compromise the DNS server [7] and corrupt DNS operations. While possibly more rare than physical failures, incorrect behavior is potentially much more serious, as faulty responses can be cached for long periods of time, and since a single faulty DNS server may send incorrect results to many clients (e.g., a single DNS root name server services on average 152 million queries per hour, to 382 thousand unique hosts [27]). With increasing deployments of physical redundancy and fast-failover technologies, software errors and vulnerabilities stand to make up an increasingly large source of DNS problems in the future.

Approaches to achieving diversity: Our approach leverages diverse replicas to recover from bugs. There are a wide variety of ways diversity could be achieved, and our architecture is amenable to several alternatives: the execution environment could be made different for each instance (e.g., randomizing layout in memory [10]), the data/inputs to each instance could be manipulated (e.g., by ordering queries differently for each server), and the software itself could be diverse (e.g., running different DNS implementations). For simplicity, in this paper we focus on software diversity. Software diversity has been widely used in other areas of computing, as diverse instances of software typically fail on different inputs [10, 12–14, 18, 28].

To estimate the level of diversity achieved across different DNS implementations, we performed static code analysis of nine popular DNS implementations (listed in the column headings of Figure 1b). First, to evaluate code diversity, we used *MOSS*, a tool used by a number of universities to detect student plagiarism of programming assignments. We used MOSS to gauge the degree to which code is shared across DNS implementations and versions. Second, to evaluate fault diversity, we used *Coverity Prevent*, an analyzer that detects programming errors in source code. We used Coverity to measure how long bugs lasted across different versions of the same software. We did this by manually investigating each bug reported by Coverity Prevent, and checking to see if the bug existed in other versions of the same software. Our results are shown in Figure 1. We found that most DNS implementations are diverse, with no code bases sharing more than one bug, and only one pair of code bases achieving a MOSS score of greater than 2% (Figure 1b). Operators of our system may wish to avoid running instances that achieve a high MOSS score, as bugs/vulnerabilities may overlap more often in implementations that share code. Also, we found that while implementation errors can persist for long periods across different versions of code, code after a major rewrite (e.g., BIND versions 8.4.7 and 9.0.0 in Figure 1a) tended to have different bugs. Hence, operators of our system may wish to run multiple versions of the same software in parallel to recover from bugs, but only versions that differ substantially (e.g., major versions).

3. DESIGN

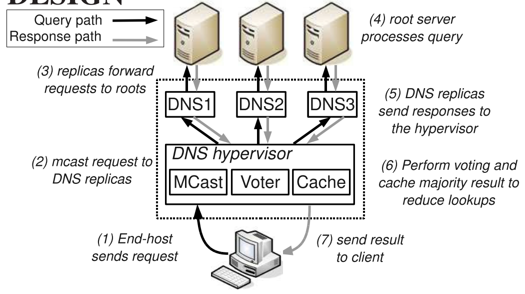

Query path
Response path
(4) root server processes query
(3) replicas forward requests to roots
DNS1
DNS2
DNS3
(5) DNS replicas send responses to the hypervisor
DNS hypervisor
(2) mcast request to DNS replicas
MCast
Voter
Cache
(6) Perform voting and cache majority result to reduce lookups
(1) End-host sends request
(7) send result to client

Figure 2: Design of DNS hypervisor.

In this section we describe the details of the design of our DNS service, which uses diverse replication to improve resilience to Byzantine failures. Our overall architecture is shown in Figure 2. Our design runs multiple replicas of DNS software atop a *DNS hypervisor*. The DNS hypervisor is responsible for mediating inputs and outputs of the DNS replicas, to make them collectively operate like a single DNS server. Our design interacts with other DNS servers using the standard DNS protocol to simplify deployment. The hypervisor is also responsible for masking bugs by using a simple voting procedure: if one replica produces an incorrect result due to a bug, or due to the fact that it is compromised by an attacker, or if it crashes, and if the instances are sufficiently diverse, then it is likely that another replica will remain available to drive execution. There are a few design choices related to DNS replicas that may affect the DR-DNS operations.

1. How many replicas to run (r)? To improve resilience to faults, the hypervisor can spawn additional replicas. Increasing the number of replicas can improve resilience, but

	8.3.0	**8.4.0**	**8.4.7**	**9.0.0**	**9.2.0**	**9.3.0**	**9.4.0**	**9.5.0**	**9.6.0**
8.3.0	200 (100%)								
8.4.0	181 (83%)	198 (100%)							
8.4.7	109 (82%)	118 (90%)	123 (100%)						
9.0.0	2 (5%)	2 (5%)	2 (5%)	108 (100%)					
9.2.0	0 (16%)	0 (15%)	0 (15%)	63 (34%)	135 (100%)				
9.3.0	0 (13%)	0 (14%)	0 (14%)	55 (26%)	119 (58%)	143 (100%)			
9.4.0	0 (12%)	0 (12%)	0 (13%)	24 (22%)	65 (50%)	76 (68%)	116 (100%)		
9.5.0	0 (11%)	0 (12%)	0 (12%)	18 (21%)	50 (47%)	59 (63%)	93 (76%)	98 (100%)	
9.6.0	0 (0%)	0 (0%)	0 (0%)	18 (23%)	47 (38%)	55 (53%)	89 (65%)	94 (71%)	110 (100%)

(a)

	BIND	**djbdns**	**MyDNS**	**NSD**	**PowerDNS**	**RBLDNSD**	**Unbound**	**Posadis**	**dnsjava**
BIND	110 (100%)								
djbdns	0 (0%)	2 (100%)							
MyDNS	0 (0%)	0 (0%)	81 (100%)						
NSD	0 (0%)	0 (0%)	0 (0%)	10 (100%)					
PowerDNS	0 (0%)	0 (0%)	0 (0%)	1 (2%)	11 (100%)				
RBLDNSD	0 (0%)	0 (0%)	0 (0%)	0 (0%)	0 (0%)	5 (100%)			
Unbound	0 (0%)	0 (0%)	0 (1%)	1 (2%)	1 (2%)	0 (0%)	30 (100%)		
Posadis	0 (0%)	0 (0%)	0 (0%)	0 (0%)	0 (0%)	0 (0%)	0 (0%)	6 (100%)	
dnsjava	0 (0%)	0 (0%)	0 (0%)	0 (0%)	0 (0%)	0 (0%)	0 (0%)	0 (0%)	6 (100%)

(b)

Figure 1: Number of overlapping bugs across code bases, with MOSS scores given in parenthesis, for (a) different versions of BIND (b) latest versions of different code bases. We find a high correlation between MOSS score and bug overlap.

incurs additional run time overheads (CPU, memory usage). In addition, there may be diminishing returns after a point. For example, we were only able to locate nine diverse copies of DNS software, and hence running more than that number of copies would not attain benefits from increased software diversity (though data diversity techniques may be applied, by manipulating inputs and execution environment of multiple replicas of the same software code base [10, 13]). Similarly, the hypervisor can kill or restart a misbehaving replica. A replica is misbehaving if it regularly produces different output than the majority result or if it crashes. In this case, the hypervisor first restarts the replica and if the problem persists, then the replica is killed and a new replica is spawned. This new replica may have different software or configuration.

2. How to select software that run in replicas? In order to increase the fault tolerance, DR-DNS administrators should choose diverse DNS implementations to run in replicas. For instance, using the same software with minor version changes (e.g., BIND 9.5.0 and BIND 9.6.0) in replicas should be avoided since those two versions will be likely to have common bugs. Instead, different software implementations (e.g., BIND and PowerDNS) or the same software implementation with major version changes (e.g., BIND 8.4.7 and BIND 9.6.0) are more suitable to run in replicas.

3. How to configure the replicas? Each DNS replica is independently responsible for returning a result for the query, though due to implementation and configuration differences, each replica may use a different procedure to achieve the result. For example, some replicas may perform iterative queries, while others perform recursive queries. To determine the result to send to the client, the DNS replicas may either recursively forward the request towards the DNS root, or may respond immediately (if they are authoritative, or have the response for the query cached). Furthermore, different cache sizes can affect the response times of replicas. For instance, a query can be cached in a replica, whereas another replica with a smaller cache may have to do a lookup for the same query.

4. How to select upstream DNS servers for replicas? Upstream DNS servers should be selected such that the possibility of propagating an incorrect result to the client is minimized. For instance, if all replicas use the same upstream DNS server to resolve the queries and if this upstream DNS server produces an incorrect result, then this incorrect result will be propagated to the end-host. However, one can easily configure replicas to select diverse upstream DNS servers that in result protects the end-users from misbehaving upstream DNS servers. External replication and path diversity techniques are further discussed in Section 6.

The hypervisor has a more complex design than replicas and it includes multiple modules: *Multicast, Voter* and *Cache*. Upon receiving an incoming query from the end-host, the hypervisor follows multiple steps. First, the *Multicast* module replicates the incoming query from the end-host and forwards the replicated queries to DNS replicas. Next, the *Voter* module waits for a set of answers received from the DNS replicas and then it generates the best answer depending on the voting scheme. For instance, a simple majority voting scheme selects the most common answer and returns it to the end-host. Finally, the answer is stored in the cache. The *Cache* module is responsible for storing the answers to common queries to reduce the response time. If the cache already has the answer to the incoming query of the end-host, then DR-DNS directly replies the answer without any further processing.

To mediate between the outputs of replicas, we use a simple *voting* scheme, which selects the majority result to send to downstream DNS/end-host clients. We propose a single voting procedure with several tunable parameters:

How long to wait (t, k)? Each replica in the system may take different amounts of time to respond to a request. For example, a replica may require additional processing time: it may be due to a less-efficient implementation, because it does not have the response cached and must perform a remote lookup, or because the replica is frozen/locked-up and not responding. To avoid waiting for an arbitrary amount of time, the voter only waits for a maximum amount of time t before continuing, and is allowed to return the majority early when k replicas return their responses.

Even though DR-DNS uses the simple majority voting scheme as default, a different voting scheme can be selected by the administrator. There are three main voting schemes DR-DNS currently supports: *Simple Majority Voting, Weighted Majority Voting*, and *Rank Preference Majority Voting*. Note that a DNS answer may include multiple ordered IP addresses. The end-host usually tries to communicate with the first IP address in the answer. The second IP address is used only if the first one fails to reply. Similarly the third address is used if the first two fails, and so on.

Simple Majority Voting: In this voting scheme, the ranking of IP addresses in a given DNS answer is ignored. IP addresses seen in majority of the replica answers win regardless of the ordering in replica answers. The final answer, however, orders the majority IP addresses according to their final counts. This voting scheme is a simplified version of the weighted majority voting scheme with all weights being equal to one.

Weighted Majority Voting: This voting scheme is based on the simple majority voting. The main difference of this voting scheme is that replicas have weights affecting the final result proportional to their weights. Replicas with more weights contribute more to the final result. Weights can be determined dynamically, or they can be assigned by the administrator statically in the configuration file. A dynamic weight of a replica is increased if the replica answer and the final answer has at least one common IP address. Otherwise, the replica is likely to have an incorrect result and its weight is decreased. In the static approach, the administrator may prefer to assign static weights to replicas. For instance, one may want to assign a larger weight to the replica using latest version of the same software compared to replicas using older versions. Similarly, an administrator may trust replicas using well-known software such as BIND more than replicas using other DNS software. The dynamic approach can adjust to transient buggy states much better than the static approach, but it includes an additional performance cost. Finally, a hybrid approach is also possible where each replica has two weights: a static and a dynamic weight. As a result, static weight is assigned by the administrator, whereas the dynamic weight is adjusted as DR-DNS processes queries.

Rank Preference Majority Voting: This voting scheme is also based on the simple majority voting. In the simplest rank preference voting, the IP addresses are weighted based on their ordering in the DNS answer. For instance, the first IP address in a replica answer is weighted more than the second IP address in the same answer. The final answer is generated by applying simple majority voting on the cumulative weights of IP addresses.

4. IMPLEMENTATION

To better understand the practical challenges of our design, we built a prototype implementation in Java, which we refer to as "Diverse Replica DNS" (DR-DNS). We had several goals for the prototype. First, we would like to ensure that the multiple diverse replicas are *isolated*, so that incorrect behavior/crashes of one replica do not affect performance of the other replicas. To achieve this, the DNS hypervisor runs each instance within its own process, and uses socket communication to interact with them. Second, we wanted to eliminate the need to modify the code of existing DNS software implementations running within our prototype. To do this, our hypervisor's voter acts like a DNS proxy, by maintaining a separate communication with each running replica and mediating across their outputs. In addition, we wanted our design to be as simple as possible, to avoid introducing potential for additional bugs and vulnerabilities that may lead to compromising the hypervisor. To deal with this, we focused on only implementing a small set of basic functionality in the hypervisor, relying on the replicas to perform DNS-specific logic. Our implementation consisted of 2,391 lines of code, with 1,700 spent on DNS packet processing, 378 lines on hypervisor logic including caching and voting, and the remaining 313 lines on socket communication. (by comparison, BIND has 409,045 lines of code, and the other code bases had 28,977-114,583 lines of code). Finally, our design should avoid introducing excessive additional traffic into the DNS system, and respond quickly to requests. To achieve this, our design incorporates a simple cache, which is checked before sending requests to the replicas. Our cache implementation uses the Least Recently Used (LRU) eviction policy.

On startup, our implementation reads a short configuration file describing the location of DNS software packages on disk, spawns a separate process corresponding to each, and starts up a software instance (replica) within each process. Each of these software packages must be configured to start up and serve requests on a different port[1]. The hypervisor then binds to port 53 and begins listening for incoming DNS queries. Upon receipt of a query, the hypervisor checks to see if the query's result is present in its cache. If present, the hypervisor responds immediately with the result. Otherwise, it forwards a copy of the query to each of the replicas. The hypervisor then waits for the responses, and selects the majority result to send to the client. To avoid waiting arbitrarily long for frozen/deadlocked/slow replicas to respond, the hypervisor waits no longer than a timeout (t) for a response. Note each replica's approach to processing the query may be different as well, increasing potential for diversity. For example, one replica may decide to iteratively process the query, while others may perform recursive lookups. In addition, different implementations may perform different caching strategies or have different cache sizes, and hence one copy may be able to satisfy the request from its cache while another copy may require a remote lookup. Regardless, the responses are processed by the hypervisor's voter to agree on a common answer before returning the result to the client.

[1]As part of future work, we are investigating use of virtual machine technologies to eliminate this requirement.

Our implementation has three main features to achieve high scalability, fast response and correctness. First, DR-DNS is implemented using threads with a thread pool. Upon start up, DR-DNS generates a thread pool including the threads that are ready to handle incoming queries. Whenever a query is received, it is assigned to a worker thread and run in parallel to other queries. The worker is responsible for keeping all the state information about the query including the replica answers. After the answer to the query is replied, the worker thread returns to the pool and waits for a new query. High scalability in our implementation can be reached by increasing the size of the thread pool as the load on the server increases. Second, DR-DNS is implemented in an event-driven architecture. The main advantage of the event-driven architecture is that it provides flexibility to process an event without any delay. In our implementation, almost all events related to replicas are time critical and need to be processed quickly to achieve fast response time. Finally, our hypervisor implementation consistently checks replicas for possible misbehavior. The replica answers are regularly checked against the majority result to notice any misbehavior to achieve high correctness.

5. REPLICATION WITHIN A SINGLE NODE

Setup: To study performance under heavy loads, we replayed traces of DNS requests collected at a large university (the University of Illinois at Urbana-Champaign (UIUC), which has roughly 40,000 students) against our implementation (DR-DNS) running on a single-core 2.5 GHz Pentium 4. The trace contains two days of traffic, corresponding to 1.7 million requests. Since some of the DNS software implementations we use make use of caches, we replay 5 minutes worth of trace before collecting results, as we found this amount of time eliminated any measurable cold start effects. We configure DR-DNS to run four diverse DNS implementations, namely: BIND version 9.5.0, PowerDNS version 3.17, Unbound version 1.02, and djbdns version 1.05. We run each replica with a default cache size of 32MB. Some implementations resolve requests iteratively, while others resolve recursively, and we do not modify this default behavior. Since modeling bug behavior is in itself an extremely hard research topic, for simplicity we consider a simple two-state model where a DNS server can be either in a *faulty* or *non-faulty* state. When faulty, all its responses to requests are incorrect, and the interarrival times between faulty states is sampled from a Poisson distribution with mean rate $\lambda_{nf} = 100000$ milliseconds. The duration of faulty states is also sampled from a Poisson distribution with mean rate $\lambda_f = \mu * \lambda_{nf}$. While for traditional failures μ is on the order of 0.0005 [20], to stress test our system under more frequent bugs (where our system is expected to perform more poorly), we consider of $\mu = 0.01$, $\mu = 0.003$, and $\mu = 0.001$.

Metrics: There are several benefits associated with our approach. For example, running multiple copies can improve resilience to Byzantine faults. To evaluate this, we measure the *fault rate* as the fraction of time when a DNS server is generating an incorrect output. At the same time, there are also several costs. For example, it may slow response time, as we must wait for multiple replicas to finish computing their results. To evaluate this, we measure the *processing delay* of a request through our system. In this section, we quantify the benefits (Section 5.1) and costs (Section 5.2) of our design.

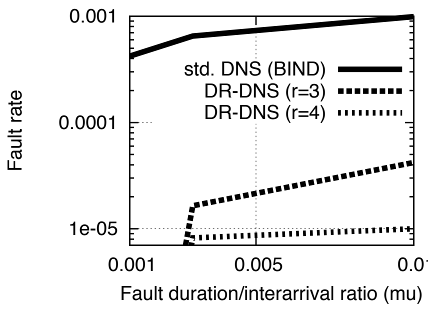


Figure 3: Effect of μ on fault rate, with t fixed at 4000ms.

5.1 Benefits

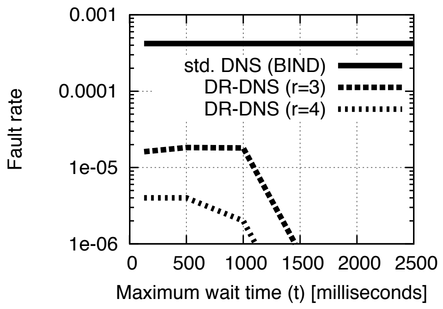


Figure 4: Effect of timeout on fault rate, with μ fixed at 0.001.

The primary benefit of our design is in improving resilience to Byzantine behavior. However, the precise amount of benefit achieved is a function of several factors, including how often Byzantine behavior occurs, how long it tends to last, the level of diversity achieved across replicas, etc. Here, we evaluate the amount of benefit gained from diverse replication under several different workloads.

First, using λ_f and λ_{nf} we measured the fraction of buggy responses returned to clients (i.e., the *fault rate*). In particular, we vary $\mu = \lambda_f / \lambda_{nf}$. For simplicity, since performance of DR-DNS is a function primarily of the ratio of these two values, we can measure performance as a function of this ratio. We found that DR-DNS reduces fault rate by multiple orders of magnitude when run with $\mu = 0.0005$. To evaluate performance under more stressful conditions, we plot in Figure 3 performance for higher ratios. We find that even under these more stressful conditions, DR-DNS reduces fault rate by an order of magnitude. We find a similar result when we vary the timeout value t, as shown in Figure 4.

Our system also can leverage spare computational capacity to improve resilience further. It does this by running additional replicas. We evaluate the effect of the number of replicas on fault rate in Figures 3 and 4. As expected, we find that increasing the number of replicas reduces fault rate. For example, when $\mu = 0.001$ and $t = 1000$, running one additional replica (increasing $r = 3$ to $r = 4$) reduces

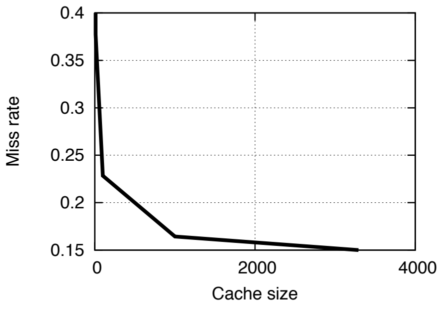


Figure 5: Amount of memory required to achieve desired hit rate.

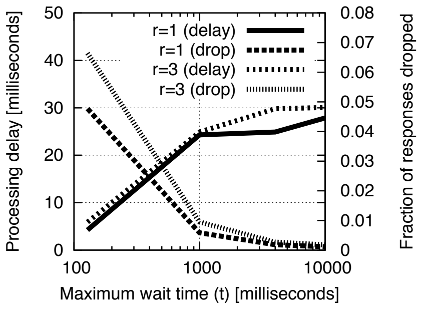


Figure 7: Effect of timeout on reducing delay.

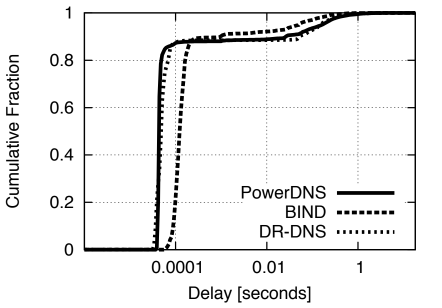


Figure 6: Amount of delay required to process requests.

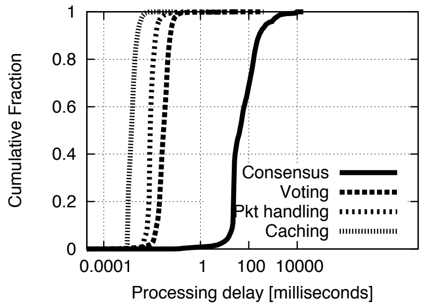


Figure 8: Microbenchmarks showing most of delay is spent waiting for replicas to reach consensus.

fault rate by a factor of eight.

5.2 Costs

First, DNS implementations are often configured with large caches to reduce request traffic. Our system increases request traffic even further, as it runs multiple replicas, which do not share their cache contents. To evaluate this, we measured the amount of memory required to achieve a certain desired hit rate in Figure 5. Interestingly, we found that reducing cache size to a third of its original size (which would be necessary to run three replicas) did not substantially reduce hit rate. To offset this further, we implemented a *shared cache* in DR-DNS's DNS hypervisor. To improve resilience to faulty results returned by replicas, DR-DNS's cache periodically evicts cached entries. While this increases hypervisor complexity slightly (adds an additional 52 lines of code), it maintains the same hit rate as a standalone DNS server.

Second, our design imposes additional delay on servicing requests, as it must wait for the multiple replicas to arrive at their result before proceeding. To evaluate this, we measured the amount of time it took for a request to be satisfied (the round trip time from a client machine back to that originating client). Figure 6 plots the amount of time to service a request. We compare a standalone DNS server running BIND with DR-DNS running $r = 3$ copies (BIND, PowerDNS, and djbdns). We find that BIND runs more quickly than PowerDNS, and DR-DNS runs slightly more slowly than PowerDNS. This is because in its default configuration, DR-DNS runs at the speed of the slowest copy, as it waits for all copies to respond before proceeding. To mitigate this, we found that increasing the cache size can offset any additional delays incurred by processing.

An alternate way to reduce delay is to vary t (to bound the maximum amount of time the voter will wait for a replica to respond) or k (to allow the voter to proceed when the first k replicas finish processing). As one might expect, we found that increasing k or increasing t both produce a similar effect: increasing them reduces fault rate, but increases delay. However, we found that manipulating t provided a way to bound worst-case delay (e.g., to make sure a request would be serviced within a certain time bound), while manipulating k provided a worst-case resilience against bugs (e.g., to make sure a response would be voted upon by at least k replicas). Also, as shown in Figure 7, we found that making t too small increased the number of dropped requests. This happens because, if no responses from replicas are received before the timeout, DR-DNS drops the request (we also considered a scheme where we wait for at least one copy to respond, and achieved a reduced drop rate at the expense of increased delay).

To investigate the source of delays in DR-DNS, we performed microbenchmarking. Here, we instrument DR-DNS with timing code to measure how much time is spent handling/parsing DNS packets, performing voting, checking the local cache, and waiting for responses from remote DNS servers. Figure 8 shows that the vast majority of request processing time is spent on waiting for the replicas to finish communicating with remote servers and to achieve consensus. This motivates our use of k and t: since these parameters control the amount of time required to achieve consensus, they provide knobs that allow us to effectively control delay (or to trade it off against fault rate).

Under heavy loads, we found that DR-DNS dropped a slightly larger number of requests than a standalone DNS server (0.31% vs. 0.1%). Under moderate and light loads, we found DR-DNS dropped fewer requests than a standalone DNS server (0.004% vs. 0.036%). This happens because there is some small amount of loss between DR-DNS and the remote root servers, and since like other schemes that replicate queries [23], our design sends multiple copies of a request, it can recover from some of these losses at the added expense of additional packet overhead.

6. REPLICATION ACROSS NODES

Our work so far has focused on *internal replication* – running multiple DNS replicas within a single host. However, the distributed nature of the DNS hierarchy means that there are often multiple remote DNS servers that can respond to a request. This provides the opportunity for DR-DNS to leverage *external replication* as well. Hence, in order to increase the reliability of the whole DNS query resolution process, we use the existing DNS hierarchy and redundancy as another form of diversity. In particular, we extend the DR-DNS design to allow its internal DNS replicas to send queries to multiple diverse upstream DNS servers and apply voting for the final answer. *Path diversity*, the selection of the diverse upstream DNS servers, can be considered as leveraging diversity across upstream DNS servers. While this approach presents some practical challenges, we present results to indicate the benefits of maintaining and increasing diversity in the existing DNS hierarchy. The rest of the section is organized as follows. Section 6.1 provides the design extensions of DR-DNS to support path diversity. Section 6.2 presents the benefits and costs of path diversity. Finally, Section 6.3 discusses the path diversity in the existence of CDNs and DNS load balancing.

6.1 Leveraging path diversity in DR-DNS

We extend the DR-DNS design to leverage path diversity in the DNS hierarchy. In the extended DR-DNS design each internal DNS replica (1) sends replicated queries to multiple diverse upstream DNS servers and (2) applies voting on the received answers. Hence, we extended each internal DNS replica with a *replica hypervisor*, i.e. a DNS hypervisor without a cache. The DNS hypervisor already has a Multicast module (MCast) to replicate the queries and Voter module to apply majority voting on the received answers. In this case, we disabled the caches of replica hypervisors since DNS replicas include their own caches. Whenever a DNS replica wants to send a query to upstream DNS servers, it simply sends the query to its replica hypervisor. Then, the multicast module in the replica hypervisor replicates the query and forwards copies to selected upstream DNS servers. Upon receiving answers, the voter module simply applies majority voting on the answers and replies to its DNS replica with the final answer.

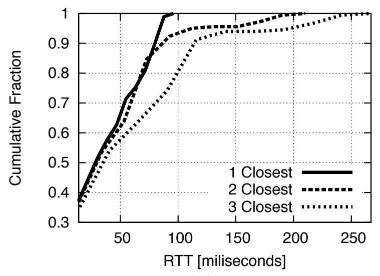
6.2 Benefits and Costs

The primary benefit of our design extension is in improving resilience to errors that can occur in any DNS servers involved in the query resolution. However, the amount of exact benefit gained depends on the level of diversity achieved across upstream DNS servers. To increase the reliability of DNS query resolution process, one needs to avoid sending queries to upstream DNS servers that share software vulnerabilities. Hence, we select the upstream DNS servers with either different software implementations (e.g., BIND and PowerDNS) or the same software implementation with major version changes (e.g., BIND 8.4.7 and BIND 9.6.0). One can also select upstream DNS servers running different operating systems (e.g., Windows or Linux).

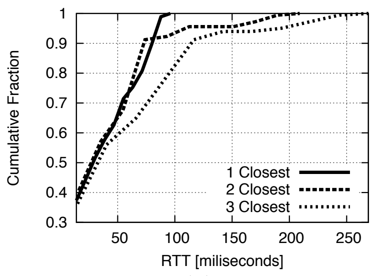
To measure diversity of the existing DNS infrastructure, we used two open-source fingerprinting tools: (1) *fpdns*, a DNS software fingerprinting tool [5], and (2) *nmap*, an OS fingerprinting tool [6]. fpdns is based on borderline DNS protocol behavior. It benefits from the fact that some DNS implementations do not offer the full set of features of DNS protocol. Furthermore, some implementations offer extra features outside the protocol set, and even some implementations do not conform to standards. Given these differences among implementations, fpdns sends a series of borderline queries and compares the responses against its database to identify the vendor, product and version of the DNS software on the remote server. The nmap tool, on the other hand, contains a massive database of heuristics for identifying different operating systems based on how they respond to a selection of TCP/IP probes. It sends TCP packets to the hosts with different packet sequences or packet contents that produce known distinct behaviors associated with specific OS TCP/IP implementations.

First, we collected a list of 3,000 DNS servers from the DNS root traces [4] on December 2008 and probed these DNS servers to check their availability from a client within the UIUC campus network. Then, we eliminated the non-responding servers. Second, we identified the DNS software and OS version of each available server with fpdns and nmap tools. This gives us a list of available DNS servers with corresponding DNS software and OS versions. One can easily select diverse upstream DNS servers from this list. However, careless selection comes with major cost: increased delay due to forwarding queries to distant upstream DNS servers compared to closest local upstream DNS server. Hence, one needs to select diverse upstream DNS servers that are close to the given host to minimize the additional delay. Here, we propose a simple selection heuristic: for a given host, we first find the top k diverse DNS servers which have the longest prefix matches with the host IP address. This results in k available DNS servers topologically very close to the host. Then, we use the King delay estimation methodology [15] to order these DNS servers according to their computed distance from the host. For practical purposes, we have used $k = 5$ in our experiments. Finally, to evaluate the additional delay, we first collected a list of 1000 hosts from [3]. Then, for each host in this list we measured the amount of extra time needed to use multiple diverse upstream DNS servers. Figures 9a (DNS software diversity) and 9b (OS diversity) plot the amount of total time to service the queries as additional diverse upstream DNS servers are accessed.

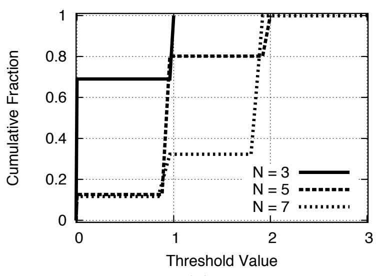
The results show that BIND is the most common DNS software among DNS servers we analyzed (69.8% BIND v9.x, 10% BIND v8.x). We also found that OS distribution among DNS servers is more balanced: 54% Linux and 46% Windows. Even though the software diversity among public DNS servers should be improved, the results indicate that current degree of diversity is sufficient for our reliability purposes. However, there is a delay cost in using multiple upstream DNS servers since we have to wait for all answers of the upstream DNS servers. This extra delay is shown in Figures 9a and 9b. We found that with an average of 26ms delay increase, we can use additional upstream DNS servers with



(a)



(b)



(c)

Figure 9: (a) Achieving diversity may require sending requests to more distant (higher-latency) DNS servers. Effect of DNS software diversity on latency inflation. (b) Effect of OS software diversity on latency inflation. (c) Number of failures that can be masked with N, the number of upstream DNS servers.

diverse DNS software to increase the reliability. Similarly, upstream DNS servers with diverse OS software can be used with an average of 19ms extra delay. We found that we can use OS diversity with a smaller overhead since OS distribution among DNS servers is more balanced. We conclude that DR-DNS extensions to use path diversity improves the reliability and protects the end users from software bugs and failures of upstream DNS servers. Moreover, the average delay cost is small and can be tolerated by the end users. Finally, our design increases the traffic load on upstream DNS servers, and this component of DR-DNS may be disabled if needed. However, we believe that the increasing severity of DNS vulnerabilities and software errors, coupled with the reduced costs of multicore technologies making computational and processing capabilities cheaper, will make this a worthwhile tradeoff.

6.3 Effect of Content Distribution Networks

Content distribution networks (CDNs) deliver content to end-hosts from geographically distributed servers using the following procedure. First, a *content provider* provide the content to a CDN. Next, the CDN replicates the content in replicas, multiple geographically distributed servers. Finally, an end-host requesting the content is redirected to one of the replicas instead of the original content provider. There are numerous advantages of CDNs: scalability, load balancing, high performance, etc. Some CDNs use *DNS redirection* technique to redirect the end-hosts to the best available CDN server for content delivery. Therefore, the CDN replica providing the content to the end-host may change dynamically depending on a few parameters including the geographic location of the end-host, network conditions, the time of the day and the load on the CDN replicas [1, 26]. As a result, a specific end-host may receive different DNS answers to the same query in subsequent requests. Hence, one might ask the question: *How does the existence of CDNs affect DR-DNS?*

DR-DNS applies majority voting to multiple DNS answers where each DNS answer includes a set of ordered IP addresses. In the existence of CDNs, DNS answers include IP addresses of CDN replicas which can deliver the content efficiently. Therefore, two DNS answers to the same query may not have any common IP addresses. This results in no winning IP set after the majority voting in DR-DNS. However, in this case DR-DNS cannot make any final decision and simply returns all IP addresses to the end-host. As a rule, DR-DNS returns all IP addresses from the DNS answers if it fails to find the majority set. Note that this approach still works *correctly* since *any* of the returned IP addresses will direct the client to a valid CDN server, and DR-DNS ensures that one of those IP addresses is always returned. However, DR-DNS heavily relies on the results of majority voting to improve the reliability. To evaluate how CDNs affect the reliability of DR-DNS, we measured the variation in DNS answers from Akamai, a well known CDN.

6.3.1 Effect of geographic location

CDNs use DNS redirection technique to redirect the end-hosts to the best available replicas. In DNS redirection, the end-host's query is handled by the DNS server authoritative for the requested domain, which is controlled by the CDN to return IP addresses of CDN replicas from which the content can be delivered most efficiently. CDN replicas for content delivery is chosen dynamically depending on the location of the end-host. For instance, an end-host located in New York may be more likely to be redirected to a replica in New Jersey rather than a replica in Seattle. Hence, in the existence of CDNs, DNS answers heavily depend on the location of the upstream DNS server. Two geographically distant upstream DNS servers will be likely to return different IP sets in the DNS answers to the same query. However, DR-DNS relies on the majority voting that elevates the common IP addresses in the returned DNS answers to improve reliability. To understand how often DR-DNS cannot do majority voting in the existence of CDNs, we carried out the following experiment. First, we selected the top 1000 most popular worldwide domains from [2] to use as queries since many content providers are in this list. Even though using top domains as queries results in biased measurements, it helps us to get an upper bound for the worst case. Next, for each query we randomly selected $N = 3, 5, 7$ upstream DNS servers from (1) the same state (Louisiana), (2) same country (USA) and (3) different countries. For the third experiment, we selected the countries from distinct continents (USA, Brazil, UK, Turkey, Japan, Australia, South Africa) to again evaluate the worst case. Table 1 shows the ratio of top domain queries where DR-DNS cannot find the majority set.

We found that CDNs affect the majority voting more if the selected upstream DNS servers are geographically distributed around the world. The results also show that CDN effects can be minimized in DR-DNS by selecting upstream DNS servers from a smaller region. For instance, selecting upstream DNS servers from the same state guarantees

	$N = 3$	$N = 5$	$N = 7$
State (Louisiana)	0.3%	0.7%	0.8%
Country (USA)	1.0%	2.0%	1.7%
World	1.6%	2.4%	2.0%

Table 1: The ratio of top domain queries where majority voting fails. N is the number of upstream DNS servers.

that DR-DNS improves the reliability of more than 99% of the queries. The main conclusion is that one should choose upstream DNS servers close to end-hosts for better reliability. Moreover, the heuristic that we developed in the previous section for path diversity chooses diverse upstream DNS servers close to the end-host, so DR-DNS already minimizes CDN effects.

	$N = 3$	$N = 5$	$N = 7$
USA - Top Domains	1.0%	2.0%	1.7%
USA - UIUC Trace	0.6%	0.9%	0.7%

Table 2: The ratio of top domain queries that majority voting fails, for USA-located hosts. The UIUC trace contains less queries to CDN clients. N is the number of upstream DNS servers.

Next, to obtain more realistic results, we repeated the same experiment with 1000 queries randomly selected from the UIUC primary DNS server trace. Table 2 shows that DR-DNS is less affected from CDNs in the UIUC trace.

6.3.2 Effect of number of upstream DNS servers

Next, we studied how the control overhead and the resilience in DR-DNS changes as we increase the number of upstream DNS servers. We found that control overhead increased linearly with the number of simultaneous requests, as expected. To evaluate the resilience, we performed the following experiment: we repeatedly send a random DNS query to multiple servers, and look at their answers. In some cases, the IP addresses in DNS answers may differ due to CDNs. If the majority voting fails, then DR-DNS doesn't improve the reliability. To evaluate performance, we then count these cases. Majority voting finds a winning IP set if more than half of the upstream DNS servers agree on at least one IP address. Let N be the number of upstream DNS servers DR-DNS queries simultaneously. Then, the minimum number of upstream servers that need to agree for the majority result is $N_{min} = \lfloor \frac{N}{2} \rfloor + 1$. For a given query, let C be the number of upstream DNS servers that agrees on the winning IP set (majority voting succeeds). Since there is a winning IP set, $C >= N_{min}$. Now, we define the threshold $T = C - N_{min}$ to measure how many extra upstream DNS servers agreed on the majority set. Note that if $T = 0$, then the majority result is agreed upon by N_{min} number of upstream DNS servers. In this case, if one server that contributes to the majority result becomes buggy, then majority voting fails. However, if $T = N - N_{min}$ is at maximum value (all upstream DNS servers agree on the winning IP set), then to fail in majority voting, $N - N_{min} + 1$ upstream DNS servers need to become buggy simultaneously. Hence, to evaluate resilience, we measure the threshold T for every query. The reliability of the majority answer is directly proportional to threshold value T. Figure 9c shows the increase in reliability as we increase the number of upstream DNS servers.

Overall, we found that for most queries, DR-DNS enabled with our external replication techniques could perform majority voting to mask the bug, thereby increasing reliability. DR-DNS was unable to do majority voting for only 0.3% of the top domain queries if three upstream DNS servers are selected from the same state. While for these small number of queries it does not mask the fault, it is important to note that it performs no worse than a normal (uninstrumented) baseline DNS system even in these cases. Finally, the reliability of the majority answer can be increased by sending queries to more upstream DNS servers.

7. RELATED WORK

DNS suffers from a wide variety of problems. Reliability of DNS can be harmed through a number of ways. Physical outages such as server failures or dropped lookup packets may prevent request processing. The DNS also suffers from performance issues, which can delay responses or increase loads on servers [27]. DNS servers may be misconfigured, which may lead to cyclic dependencies between zones, or cause servers to respond incorrectly to requests [22]. Also, implementation errors in DNS code can make servers prone to attack, and can lead to faulty responses [7, 25].

Dealing with failures in DNS is certainly not a new problem. For example, DNS root zones are comprised of hundreds of geographically distributed servers, and anycast addressing is used to direct requests to servers, reducing proneness to physical failures. Redundant lookups and cooperative caching can substantially reduce lookup latencies and resilience to fail-stop failures [23, 24]. Troubleshooting tools that actively probe via monitoring points can detect large classes of misconfigurations [22]. Our work does not aim to address fail-stop failures, and instead we leverage these previous techniques, which work well for such problems.

However, these techniques do not aim to improve resilience to problems arising from implementation errors in DNS code. A vulnerability in a single DNS root server affects hundreds of thousands of unique hosts per hour of compromise [11, 27], and a single DNS name depends on 46 servers on average, whose compromise can lead to domain hijacks [25]. The DNS has experienced several recent high-profile implementation errors and vulnerabilities. As techniques dealing with fail-stop failures become more widely deployed, we expect that implementation errors may make up a larger source of DNS outages. While there has been work on securing DNS (e.g., DNSSEC), these techniques focus on authenticating the source of DNS information and checking its integrity, rather than masking incorrect lookup results. In this work, we aim to address this problem at its root, by increasing the software diversity of the DNS infrastructure.

Software diversity techniques have been used to prevent attacks on large scale networks in multiple studies. It has been shown that reliability of single-machine servers to software bugs or attacks can be increased with diverse replication [13]. In another work, diverse replication is used to protect large scale distributed systems from Internet catastrophes [18]. Similarly, to limit malicious nodes to compromise its neighbors in the Internet, software diversity is used to assign nodes diverse software packages [21]. In another work, to increase the defense capabilities of a network, the authors suggest increasing the diversity of nodes to make the network more heterogeneous [29]. To the best of our knowledge, our work is the first to directly address the root cause of implementation errors in DNS software, via the use of diverse replication. However, our work is only an early first step in this direction, and we are currently investigating a wider array of practical issues as part of future work.

8. CONCLUSIONS

Today's DNS infrastructure is subject to implementation errors, leading to vulnerabilities and buggy behavior. In this work, we take an early step towards addressing these problems with *diverse replication*. Our results show that available DNS software packages have sufficient diversity in code resulting in a minimal number of shared bugs. However, DNS software with minor version changes share most of the code base resulting in less diversity. We have also found that the number of bugs is not reduced in later versions of the same software since usually new functionality is added to software introducing new bugs. We also find that our system masks buggy behavior with diverse replication, reducing the fault rate by an order of magnitude. Increasing the number of replicas further decreases the fault rate. Our results indicate that DR-DNS runs quickly enough to keep up with the loads of a large university's DNS queries. In addition, DR-DNS can leverage redundancy in the current DNS server hierarchy (replicated DNS servers, public DNS servers, etc.). We can use this redundancy to select diverse upstream DNS servers to protect the end-host from possible errors existing in the upstream servers. Selecting a different upstream DNS server may increase response time, but our results show that a slight increase in response time enables a significant improvement in reliability. CDNs and DNS-level load balancing may result in DNS queries being resolved to different sets of IP addresses, which can limit ability of DR-DNS to mask bugs across remote servers. However, our results indicate that performance is reduced only minimally in practice, and correctness of operation is not affected.

While our results are promising, much more work remains to be done. First, we plan to design a *server-side* voting strategy, to protect the DNS root from bogus queries [27]. Also, we plan to investigate whether porting our Java-based implementation to C++ will speed request processing further. We are also currently in the process of deploying our system for use within the campus network of a large university, to investigate practical issues in a live operational network. Finally, we plan to extend our study to include many other protocols to investigate how diversity changes among protocols. This helps us to generalize our method for other protocols.

9. REFERENCES

[1] Akamai. `http://www.akamai.com`.

[2] Alexa. `http://www.alexa.com`.

[3] CAIDA. `http://www.caida.org/data/`.

[4] DNS-OARC. Domain name system operations, analysis, and research center. `http://www.dns-oarc.net`.

[5] fpdns - DNS fingerprinting tool. `http://code.google.com/p/fpdns`.

[6] Insecure org. The nmap tool. `http://www.insecure.org/nmap`.

[7] Securityfocus: Bugtraq mailing list. `http://www.securityfocus.com/ vulnerabilities`.

[8] Root nameserver (Wikipedia article). `http://en.wikipedia.org/wiki/Root_nameserver`.

[9] Bent, L., and Voelker, G. Whole page performance. In *The 7th International Web Caching Workshop (WCW)* (August 2002).

[10] Berger, E., and Zorn, B. Diehard: Probabilistic memory safety for unsafe languages. In *Programming Languages Design and Implementation* (June 2006).

[11] Brownlee, N., kc claffy, and Nemeth, E. DNS measurements at a root server. In *IEEE GLOBECOM* (November 2001).

[12] Castro, M., and Liskov, B. Practical byzantine fault tolerance. In *OSDI* (February 1999).

[13] Chun, B.-G., Maniatis, P., and Shenker, S. Diverse replication for single-machine byzantine-fault tolerance. In *USENIX ATC* (June 2008).

[14] Forrest, S., Hofmeyr, S. A., Somayaji, A., and Longstaff, T. A. A sense of self for unix processes. In *IEEE Symposium on Security and Privacy* (1996), pp. 120–128.

[15] Gummadi, K. P., Saroiu, S., and Gribble, S. D. King: Estimating latency between arbitrary Internet end hosts. In *SIGCOMM Internet Measurement Workshop* (2002).

[16] Gupta, D., Lee, S., Vrable, M., Savage, S., Snoeren, A., Vahdat, A., Varghese, G., and Voelker, G. Difference engine: Harnessing memory redundancy in virtual machines. In *OSDI* (December 2008).

[17] Jung, J., Sit, E., Balakrishnan, H., and Morris, R. DNS performance and the effectiveness of caching. In *ACM SIGCOMM* (October 2002).

[18] Junqueira, F., Bhagwan, R., Hevia, A., Marzullo, K., and Voelker, G. Surviving Internet catastrophes. In *USENIX ATC* (April 2005).

[19] Keller, E., Yu, M., Caesar, M., and Rexford, J. Virtually eliminating router bugs. In *CoNEXT* (December 2009).

[20] Markopoulou, A., Iannaccone, G., Bhattacharyya, S., Chuah, C.-N., and Diot, C. Characterization of failures in an IP backbone. In *IEEE INFOCOM* (March 2004).

[21] O'Donnell, A. J., and Sethu, H. On achieving software diversity for improved network security using distributed coloring algorithms. In *CCS '04: Proceedings of the 11th ACM conference on Computer and communications security* (New York, NY, USA, 2004), ACM, pp. 121–131.

[22] Pappas, V., Faltstrom, P., Massey, D., and Zhang, L. Distributed DNS troubleshooting. In *ACM SIGCOMM Workshop on Network Troubleshooting* (August 2004).

[23] Park, K., Pai, V. S., Peterson, L., and Wang, Z. CoDNS: Improving DNS performance and reliability via cooperative lookups. In *OSDI* (December 2004).

[24] Ramasubramanian, V., and Sirer, E. G. The design and implementation of a next generation name service for the Internet. In *ACM SIGCOMM* (August 2004).

[25] Ramasubramanian, V., and Sirer, E. G. Perils of transitive trust in the domain name system. In *Internet Measurement Conference* (October 2005).

[26] Su, A.-J., Choffnes, D. R., Kuzmanovic, A., and án E. Bustamante, F. Drafting behind Akamai (Travelocity-based detouring). In *ACM SIGCOMM* (2006).

[27] Wessels, D., and Fomenkov, M. Wow, that's a lot of packets. In *Passive and Active Measurement* (April 2003).

[28] Yumerefendi, A., Mickle, B., and Cox, L. Tightlip: Keeping applications from spilling the beans. In *NSDI* (April 2007).

[29] Zhang, Y., Vin, H., Alvisi, L., Lee, W., and Dao, S. K. Heterogeneous networking: A new survivability paradigm. In *NSPW '01: Proceedings of the 2001 workshop on New security paradigms* (New York, NY, USA, 2001), ACM, pp. 33–39.

[30] Zhou, Y., Marinov, D., Sanders, W., Zilles, C., d'Amorim, M., Lauterburg, S., and Lefever, R. Delta execution for software reliability. In *Hot Topics in System Dependability* (June 2007).

Enabling Secure VM-vTPM Migration in Private Clouds

Boris Danev, Ramya Jayaram Masti, Ghassan O. Karame and Srdjan Capkun
Department of Computer Science
ETH Zurich, Switzerland
{bdanev, rmasti, karameg, capkuns}@inf.ethz.ch

ABSTRACT

The integration of Trusted Computing technologies into virtualized computing environments enables the hardware-based protection of private information and the detection of malicious software. Their use in virtual platforms, however, requires appropriate virtualization of their main component, the Trusted Platform Module (TPM) by means of virtual TPMs (vTPM). The challenge here is that the use of TPM virtualization should not impede classical platform processes such as virtual machine (VM) migration.

In this work, we consider the problem of enabling secure migration of vTPM-based virtual machines in private clouds. We detail the requirements that a secure VM-vTPM migration solution should satisfy in private virtualized environments and propose a vTPM key structure suitable for VM-vTPM migration. We then leverage on this structure to construct a secure VM-vTPM migration protocol. We show that our protocol provides stronger security guarantees when compared to existing solutions for VM-vTPM migration. We evaluate the feasibility of our scheme via an implementation on the Xen hypervisor and we show that it can be directly integrated within existing hypervisors. Our Xen-based implementation can be downloaded as open-source software. Finally, we discuss how our scheme can be extended to support live-migration of vTPM-based VMs.

1. INTRODUCTION

Trusted Computing [1] is a set of technologies that provide hardware and software support for secure storage and software integrity protection. Its integration into virtualized computing systems [2] enables the hardware-based protection of private (sensitive) information and the detection of malicious software that aims to subvert the operation of virtualized environments. While these enhancements add an additional layer of security to the underlying data and applications [3], the use of Trusted Computing in virtual platforms raises several challenges with respect to virtualization of its hardware root of trust, the Trusted Platform Module (TPM), which provides secure storage and cryptographic operations.

ACSAC '11 Dec. 5-9, 2011, Orlando, Florida USA
Copyright 2011 ACM 978-1-4503-0672-0/11/12 ...$10.00.

In fact, there is typically a single TPM module per hardware platform. Therefore, its functionality has to be efficiently shared by the virtual machines (VM) running on the same hardware. This is typically achieved by virtual TPMs (vTPMs) that mimic the interface and functionality of the hardware TPM. One important challenge is to realize vTPMs that comply with TPM specifications while not impeding platform processes such as VM migration.

Although a number of realizations of vTPMs have been proposed [4–7], several issues remain unresolved with respect to the application of these realizations to the migration of vTPMs and their corresponding VMs (VM-vTPM). In this respect, current schemes for secure VM-vTPM migration protocols either increase the virtualized platform dependence on Privacy CA [4, 7, 8] or violate TPM usage restrictions [5]. Moreover, as far as we are aware, there is no implementation of a secure VM-vTPM migration protocol.

In this work, we consider the problem of enabling secure vTPM-based VM (VM-vTPM) migration in virtualized environments. We first extend and detail the requirements that a secure VM-vTPM migration protocol should satisfy in private cloud environments. In these environments, a central provider owns a number of virtualized servers and wishes to transfer VMs along with their corresponding vTPMs across these servers (e.g., for load balancing purposes). Given this, we identify three security requirements related to the authenticity of the migration initiator, the preservation of the trust chain among entities during migration and the confidentiality of the migration process. We argue that these requirements are sufficient to support secure VM-vTPM migration in private cloud environments.

Second, we discuss the implications of current vTPM key hierarchy designs on the efficiency and performance of the VM-vTPM migration process. More specifically, we show that existent vTPM key structures are either decoupled from their corresponding hardware TPM keys or are tightly bound to the hardware TPM. The former approaches do not benefit from the security guarantees of TPM, while the latter make the vTPM keys non-migratable according to the TPM specification and therefore required costly key regeneration at the destination after the VM-vTPM migration process. Based on these observations, we derive a novel vTPM key hierarchy that introduces an intermediate layer of keys between the TPM and vTPM and provides a logical separation of the vTPM keys according to their usage in the VM. Our proposed key hierarchy also enables vTPM key migration according to the TPM usage restrictions and minimizes the dependence of vTPM keys on Privacy CA.

In addition, we propose and analyze a secure VM-vTPM migration protocol that leverages on our vTPM key hierarchy. Our proposed protocol provides stronger security guarantees when compared to existing solutions for VM-vTPM migration. Furthermore, we implement a preliminary Xen-based prototype of our protocol and we evaluate its performance. Our implementation demonstrates that our secure VM-vTPM migration solution can be directly integrated with the open-source hypervisor Xen [9]. We note that our implementation is open source and is available for download at [10]. Finally, we discuss how our scheme can be extended to support live-migration of vTPM-based VMs.

We point out that while prior work has addressed different aspects of secure VM migration, including vTPM migration, to the best of our knowledge, this work is the first to explicitly define the requirements, propose a suitable vTPM key hierarchy and design and implement a complete VM-vTPM migration protocol.

The rest of this paper is organized as follows. In Section 2, we detail our system and attacker models and derive the corresponding security requirements. We then present a novel vTPM key hierarchy and we describe a secure migration protocol in Section 3. In Section 4, we present a feasibility study and preliminary performance results extracted from a preliminary prototype implementation using the Xen hypervisor. Finally, we overview the related work in Section 5 and conclude in Section 6.

2. PROBLEM AND SECURITY REQUIREMENTS

2.1 System Model

In this paper, we consider a setting where a cloud provider $\mathcal{P}$ possesses several (> 2) virtualized servers that are equipped with physical TPMs and wishes to *securely* migrate virtual machines (VM) among these servers (e.g., for load-balancing purposes). Here, each virtual machine interfaces with the physical TPM through a software-based virtual TPM (vTPM) (refer to Section 3.1 for further details). We assume that vTPMs do not contain hardware and hypervisor configuration information; this information (stored in the TPM) is obtained by querying the TPM. Similarly, the hardware TPM does not include any VM specific information. This procedure decouples the vTPM from hardware-specific characteristics and enables its migration.

We assume that $\mathcal{P}$ wishes to migrate a virtual machine from a source server $\mathcal{S}$ to a destination server $\mathcal{D}$. We assume that $\mathcal{S}$ and $\mathcal{D}$ are equipped with public/private key pairs that are persistently stored on their respective TPMs[1]. During the migration process, we assume that the virtual machine can be suspended on $\mathcal{S}$ before it is transferred to $\mathcal{D}$; once the transfer is completed, the virtual machine is resumed on $\mathcal{D}$.

Given this setting, we consider the problem of enabling secure migration of a VM along with its vTPM from $\mathcal{S}$ to $\mathcal{D}$. Here, several challenges need to be overcome to ensure the liveliness and soundness of the migration process, namely: *(i)* only trusted servers should execute correct VMs, *(ii)* no external entity should be able to modify/learn the contents of the VM during the migration process and *(iii)* the migration can only be initiated by trusted parties (e.g., $\mathcal{P}$ or $\mathcal{S}$ in our case). Furthermore, this entire process should not violate the trusted computing standards [1] and should be easy to integrate with current vTPM platforms/architectures.

[1]Alternatively, the private keys could be sealed with TPM-specific keys and stored on disk.

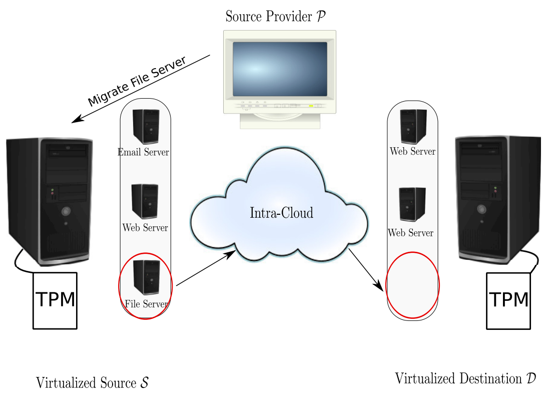

Source Provider $\mathcal{P}$
Migrate File Server
Email Server
Web Server
File Server
TPM
Intra-Cloud
Web Server
Web Server
TPM
Virtualized Source $\mathcal{S}$
Virtualized Destination $\mathcal{D}$

Figure 1: System model: a service provider $\mathcal{P}$ wishes to migrate a virtual machine from a virtualized source server $\mathcal{S}$ to a virtualized destination server $\mathcal{D}$ given some security constraints. Here, both $\mathcal{S}$ and $\mathcal{D}$ are equipped with hardware TPMs.

In what follows, we detail the attacker model and the security properties that a migration protocol should satisfy.

2.2 Attacker Model

We assume the presence of an attacker $\mathcal{A}$ that can eavesdrop, modify, insert or delete messages in the network. We assume that $\mathcal{A}$ is interested in abusing the migration protocol to increase her benefit in the network (e.g., starting her own VM, acquiring information about the transferred VM, etc.). We further assume that $\mathcal{A}$ is capable of exploiting software vulnerabilities of remote servers.

However, we point out that $\mathcal{A}$ does not have physical access to any server in the network managed by $\mathcal{P}$. In addition, we assume that $\mathcal{A}$ is computationally bounded, in the sense that she cannot forge signatures, break authentication schemes, without possessing the correct credentials.

Thus, we can safely assume that the TPM embedded in the various virtualized servers can be trusted. This trust assumption also extends to the software tool (e.g., IMA [12], HyperSafe [13]) that measures the system state for attestation using a (dynamic) root of trust (e.g., Intel TXT [14], Flicker [15]). Otherwise, little can be done to ensure that the software hosted on a given server is "authentic" and can be trusted. For simplicity and without loss of generality, we assume here that $\mathcal{A}$ cannot compromise or modify the state of software on the source and/or destination server during the migration protocol. That is, $\mathcal{A}$ can only compromise the software hosted by $\mathcal{S}$ and/or $\mathcal{D}$, either *before* the start or *after* the end of the migration process.

2.3 Requirements for Secure VM-vTPM Migration Protocols

We now present the requirements that any secure VM-vTPM migration protocol should satisfy. These requirements are sufficient for secure VM-vTPM migration and concern the parties involved in the migration as well as the communication channel over which the migration occurs. In Section 3, we also describe a protocol that fulfills them.

Requirement 1. ***(VM-vTPM Confidentiality and Integrity)***
An untrusted entity should not be able to learn any meaningful information about the VM-vTPM during the migration process. This includes the suspension, transfer and resumption of the VM-vTPM from the source to the destination. Furthermore, any modification to the VM-vTPM during the migration process should be detectable.

In addition to the basic confidentiality property, preserving the integrity of the vTPM during the migration process emerges as an important requirement for any secure VM-vTPM migration protocol. Otherwise, an adversary can convince $\mathcal{D}$ to accept a different VM image by modifying the contents of its vTPM.

Requirement 2. ***(Initiation Authenticity)***
An untrusted entity should not be able to migrate any VM-vTPM. Only $\mathcal{P}$ should be allowed to initiate the VM-vTPM migration process.

Restricting the initiation of the migration process solely to those authorized entities prevents an adversary from continuously migrating VMs across servers, thus alleviating Denial of Service (DoS) attacks against the entire system. Ensuring that only trusted parties can initiate the migration also prevents collocation attacks, where the attacker places the target VM on the same physical server together with another VM that it controls; this would create a covert-channel that might leak information about the target VM [11].

Although the initiation authenticity notion is important to prevent abuse of the migration process, it is has not been addressed, as far as we are aware, in prior work.

Requirement 3. ***(Preserving the Trust Chain)***
Only trusted servers can receive correct VMs. More specifically, (i) *trusted servers should not hold incorrect VMs-vTPMs and* (ii) *untrusted servers should not acquire correct VMs-vTPMs.*

By a correct VM, we refer to a VM which is found to be correct according to a trusted integrity measurement module (e.g., TPM-based attestation).

Ensuring the integrity of the software hosted on both $\mathcal{S}$ and $\mathcal{D}$ prior to the migration process is of paramount importance. A correct VM running in a trusted environment should not be transferred to a server that might be compromised (e.g., the VM might contain sensitive data). Similarly, a trusted server should not accept to run an incorrect VM that might have been compromised. For instance, if the hypervisor in $\mathcal{S}$ was compromised, then $\mathcal{D}$ cannot trust any protocol it establishes with $\mathcal{S}$. Note that this requirement does not address the case where an untrusted server executes incorrect VMs.

3. A SECURE VM-VTPM MIGRATION PROTOCOL

In this section, we outline an efficient solution that enables secure VM-vTPM migration. We start by providing the necessary background on TPM and vTPM keys.

3.1 vTPM Key Hierarchy

The design of the vTPM key hierarchy should provide the same functionality as the original TPM key hierarchy, i.e., allow proof of authenticity, attestation and secure storage [16]. In addition, it should comply with the TPM key usage restrictions and introduce minimal overhead during VM-vTPM migration (e.g., minimal key regeneration). In what follows, we provide a brief background on TPM functionality and keys and discuss several issues with existent vTPM key hierarchy proposals. We then introduce our vTPM hierarchy and discuss its implications on VM-vTPM migration.

Background on TPM Keys: The hardware TPM enables proofs of authenticity, attestation and secure storage based on three main cryptographic keys, namely the *Endorsement Key* (EK), the *Storage Root Key* (SRK) and the *Attestation Identity Key* (AIK). The EK is a persistent non-migratable encryption key that is used to establish the authenticity of the TPM. The use of this key for transaction authentication in the network is not recommended as it would enable TPM transaction linking. The SRK is a non-migratable encryption key that is used to protect the storage of other TPM keys outside the TPM. The AIK is an asymmetric non-migratable signing key generated inside the TPM and certified by a Certified Authority (Privacy CA). It is used as a one time key to establish authenticity of the TPM during attestation [1]. The AIK certificate proves the the AIK was created by a genuine TPM. Since it does not expose the EK, it can be safely used in network transactions without privacy concerns. The Platform Configuration Registers (PCR) are additional components used for attestation and secure storage; these components reside inside the TPM and store platform configuration measurements[2]. The latter are used either to attest the system integrity during remote attestation or seal data to particular system configurations [1].

Background on vTPM Keys: vTPM key hierarchies include keys analogous to their TPM key hierarchy counterparts. Each vTPM typically has its own virtual EK (vEK), virtual SRK ($vSRK$) which is used to protect the storage of other vTPM keys and virtual AIKs ($vAIK$s) used for platform attestation purposes.

The relationship between vTPM and TPM key hierarchies is an important design choice that needs to be taken into account in secure VM-vTPM migration. Several vTPM key hierarchy proposals completely decouple their keys from the TPM keys [4, 7]. This is achieved by obtaining the vTPM EK (vEK) and AIK ($vAIK$) credentials from a local authority. While this procedure avoids generating those keys on the platform vTPM after migration, it is not clear how it removes the need for vTPM credential regeneration. The inclusion of TPM PCRs in the certificate of a vEK to achieve VM-vTPM binding would require its frequent regeneration if TPM PCRs are periodically modified (extended) by means of dynamic system measurements [4]. All $vAIK$s obtained before the TPM PCRs changed would not be valid anymore. In addition, using a permanent vEK to prove vTPM-TPM binding during attestation [4] allows linking vTPM transactions. On the other hand, tight coupling of the vTPM and TPM (as discussed in [4]) by signing vTPM credentials

[2]These measurements often consist of hashing the state of the software running on the platform.

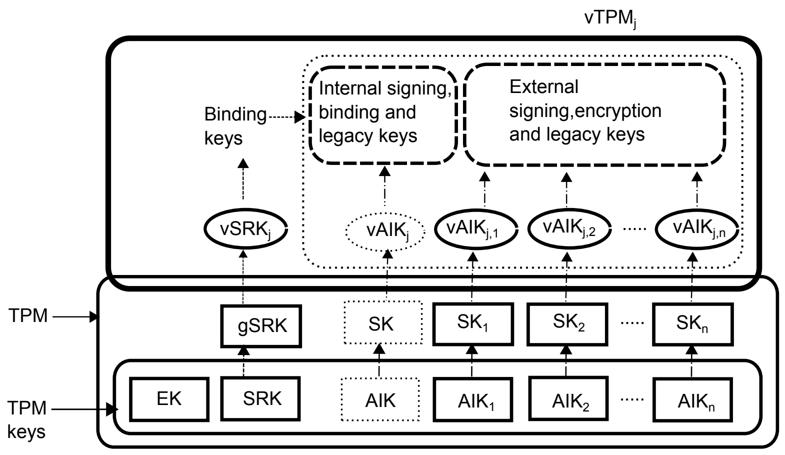


Figure 2: vTPM key set and hierarchy. Our proposed hierarchy consists of an intermediate layer of a global SRK ($gSRK$) and a set of signing keys (SKs) that connect the TPM SRK and AIKs to the vTPM $vSRK$ and $vAIK$. We also logically separate the vTPM keys into internal and external keys.

using the TPM AIK directly mandates that the corresponding keys be non-migratable and thus, requires extensive regeneration of vTPM keys on the destination platform after VM migration. The same limitation arises for the $vSRK$ if it is encrypted directly using the TPM SRK. Refer to Appendix A for more details regarding existent vTPM key hierarchy designs.

Our vTPM Key Hierarchy: In order to enable migration with minimized key regeneration after VM-vTPM migration, we propose a vTPM key hierarchy which introduces an intermediate layer of keys between the TPM and vTPM. This intermediate layer consists of one global SRK ($gSRK$) and a set of signing keys (SK) that connect the TPM SRK and AIKs to the vTPM $vSRK$ and $vAIK$ respectively (Figure 2). Even though this renders the signing keys $gSRK$ and SKs non-migratable[3], it allows the migration of the $vSRK$ and $vAIK$s and preserves the strong binding between the TPM and vTPM. Furthermore, using a separate SK with every $vAIK$ used in external communications prevents linking different vTPM transactions. We point out here that the $vSRK$ and $vAIK$ credentials can only be generated on a TPM containing the corresponding SRK and AIK. Generating the $vAIK$s on the platform itself removes the need for vEK because the authenticity of the vTPM only depends on the TPM AIK.

We further separate the vTPM keys into internal and external keys (see Figure 2). Internal vTPM keys are retained across VM migration. These include the $vSRK$ and the encryption and signing keys used only within the VM. The encryption keys are part of the $vSRK$ hierarchy and the credentials of the signing keys are signed by a $vAIK$ key linked to a TPM AIK (the key chain is shown in dotted ovals in Figure 2). Given that one such $vAIK$ could be sufficient for all internal signing, binding and legacy keys, VM-vTPM migration would incur minimal regeneration at the destination[4]. External vTPM keys are those keys used for signing and encrypting data exchanged between VMs over the network. Corresponding $vAIK$s are therefore restricted to one-time use in order to prevent vTPM transaction linking. Hence, these $vAIK$s are not part of a migrating vTPM.

[3]This is the case in order to preserve compliance with the TPM key usage restrictions

[4]We note, however, that several $vAIK$ can be used if needed.

Below we provide a summary description of our vTPM hierarchy keys and discuss the implications of this hierarchy on VM-vTPM migration:

- *vSRK*: Analogous to the TPM SRK, the vSRK protects the storage of other TPM keys. However, the storage of the vSRK itself is protected using the global SRK.

- *Global SRK (gSRK)*: This is an non-migratable asymmetric encryption key that is a direct descendant of the TPM SRK. It is used to protect the $vSRK$ of individual TPMs (by sealing) which in turn protects the other keys of their respective vTPMs (also by sealing). Creating this intermediate $gSRK$ makes the $vSRK$s migratable which would have not been possible if they were direct descendants of the TPM SRK.

- *vAIK*: Analogous to the TPM AIK, a $vAIK$ can be used to establish the authenticity of the vTPM and to sign other keys. We use a special $vAIK$ instance to sign data and/or certificates used only within the VM. This instance is transferred to the destination during vTPM migration. If a $vAIK$ signs data and/or certificates to be sent over the network, it is restricted to one time use to prevent vTPM transaction linking. Such AIKs are not part of a migrating vTPM. All $vAIK$s are linked to the TPM AIK via their own signing keys (SK).

- *Signing Keys (SKs):* These are an intermediate layer of non-migratable TPM signing keys that associate $vAIK$s with TPM AIKs. At least one SK is used for the special $vAIK$ instance (see above). Note that this SK can be common to all vTPMs on the same platform. All the other SKs are used to bind $vAIK$s to TPM AIKs intended to sign data and/or certificates to be sent over the network. SKs are not migrated during vTPM migration and therefore need to be generated on the destination platform. This entire key hierarchy is depicted in Figure 2.

Similar to most software-based vTPM solutions, our vTPM keys are stored outside the TPM and are prone to leakage and unauthorized modification. While the confidentiality of vTPM keys is protected by the $vSRK$, it is also possible to protect their integrity by the use of hash verification. This enables the detection of key modification, but does not prevent denial of service attacks by modifying the hashes themselves (on the filesystem).

During migration, the $vSRK$ of the vTPM is unsealed from the TPM using the corresponding $gSRK$ and is transferred along with other vTPM keys that are used only within the VM including the special $vAIK$ instance. At the destination, after migration, the $vSRK$ is sealed to the destination's TPM using its $gSRK$. Furthermore, the credentials for the special $vAIK$ instance are regenerated using the destination's special AIK and SK instances.

3.2 Protocol Description

Given the above vTPM key hierarchy, we proceed to presenting a possible construction of a secure VM-vTPM migration protocol (Figure 3).

Our exemplary construction mainly consists of three stages: the authentication stage, the attestation stage and the data transfer stage. In the first stage, the *authentication stage*, $\mathcal{S}$ and $\mathcal{D}$ mutually authenticate each other using their public key certificates and establish a secure channel for their subsequent communication. This can be achieved, for example, by using a non-migratable binding key that is stored on the TPM and that is bound to a secure configuration of either $\mathcal{D}$ or $\mathcal{S}$. Although this approach has clear advantages, it becomes rather costly as the message size increases; that is, the protocol will incur a prohibitively high overhead e.g., when a VM RAM is transferred from $\mathcal{S}$ to $\mathcal{D}$. In that case, a more suitable approach would be to rely on the establishment of Diffie-Hellman symmetric keys [17] between $\mathcal{S}$ and $\mathcal{D}$. For instance, this can be realized by using the TLS handshake protocol [18]. Once a session key K is established, $\mathcal{S}$ and $\mathcal{D}$ can use it to ensure the confidentiality and integrity of their communication. This can, for example, be done by concatenating each message with its hash (for integrity verification) and encrypting the result using key K (for confidentiality). Since we assume that an attacker cannot compromise the machines of $\mathcal{S}$ and $\mathcal{D}$ during the migration process, the established session keys can be stored in the system memory of both $\mathcal{S}$ and $\mathcal{D}$.

Once the authentication stage is completed, the *attestation stage* starts. This stage mainly consists of the integrity verification of both $\mathcal{S}$ and $\mathcal{D}$. In Section 3.3, we show that this verification prevents a considerable number of security threats.

To verify the integrity of $\mathcal{D}$, $\mathcal{S}$ proceeds as follows. It initiates the attestation process by sending $\mathcal{D}$ a freshly generated random nonce N_s. This would trigger a measurement module in $\mathcal{D}$ to perform a system measurement. System measurements typically include load and/or run time properties of the hypervisor [12, 13]. These properties can be measured using a number of techniques such as [19, 20].

The load time integrity of the measurement module itself can be further protected using a dynamic root of trust (like in Flicker [15]), which also provides a secure isolated run time environment. The measurement module also extends the public key certificate of $\mathcal{D}$ (or its hash) into the PCRs[5]. Given this, $\mathcal{D}$ then sends a signed copy of its PCRs (i.e., $\mathcal{D}$ sends $Sign_{AIK}(PCR \mid\mid N_s)$ signed using an AIK key obtained from a Privacy CA) containing details about the execution of the measurement module, the system configuration, its public key certificate along with a freshly generated random nonce N_s. We point out that these integrity measurements do not include any information corresponding to the contents of the VM being transferred. Instead, the integrity of the transferred VM is verified by $\mathcal{S}$ prior to migration (if any) and by $\mathcal{D}$ before resumption. $\mathcal{S}$ then verifies that the extracted PCRs correspond to those of $\mathcal{D}$ by checking the public key certificate extension into the PCRs. It then checks the validity of the AIK to verify the authenticity of $\mathcal{D}$'s TPM and $\mathcal{D}$'s PCRs to verify $\mathcal{D}$'s integrity. Similarly, $\mathcal{D}$ also verifies the integrity of $\mathcal{S}$. If these verifications pass, then the *data transfer stage* can start.

In this last stage, the actual transfer of the VM-vTPM occurs. Here, $\mathcal{D}$ sends $\mathcal{S}$ a freshly generated random nonce N_d indicating its readiness to receive the migrating VM-vTPM. $\mathcal{S}$ then transfers the contents of the VM-vTPM along with the received nonce on the established secure channel. In our construction, we require that $\mathcal{D}$ also checks the integrity of the migrated VM (for the reasoning why, refer to Section 3.3). Since it is assumed that the vTPM (or VM) queries the underlying TPM to obtain hardware and hypervisor measurement information, no separate mechanisms are required to update the vTPM with this information after migration. Finally, $\mathcal{S}$ deletes its local copy of the VM-vTPM and both $\mathcal{S}$ and $\mathcal{D}$ resume their operation.

3.3 Security Analysis

In what follows, we briefly analyze the security of our protocol construction.

The establishment of a secure channel between $\mathcal{S}$ and $\mathcal{D}$ ensures the confidentiality and integrity of all their exchanged messages. Furthermore, the use of Diffie-Hellman session keys ensures the forward security of the exchanged messages. That is, an attacker $\mathcal{A}$ cannot acquire the session key K once the migration protocol is terminated, even if it gains full control of $\mathcal{S}$ and all the exchanged messages between $\mathcal{S}$ and $\mathcal{D}$.[6] To acquire the key, $\mathcal{A}$ has to compromise $\mathcal{D}$. As such, our protocol construction satisfies the VM-vTPM confidentiality and integrity requirement (Requirement 1).

Furthermore, since the public key certificate of $\mathcal{D}$ (and $\mathcal{S}$, respectively) is extended in the PCRs during its integrity verification, $\mathcal{S}$ can ensure that the measured PCRs correspond to the physical machine of $\mathcal{D}$.[7] This prevents $\mathcal{A}$ from presenting measurements performed on *another* machine and claiming that they pertain to the machine of $\mathcal{D}$ (or $\mathcal{S}$, respectively); in this case, this misbehavior will be

[5]A variant scheme for linking the public key to the PCRs relies on the use of special TLS certificate extensions—which might, however, increase the size of trusted computing base (TCB) [21].

[6]Recall in this case that $\mathcal{S}$ can securely delete K at the end of the migration process.

[7]Linking the PCR measurements to $\mathcal{D}$ cannot be achieved solely by the use of the AIK of $\mathcal{D}$. This is because the AIK does not contain any information that could be used for identifying the entity to which it was issued (in this case, $\mathcal{S}$ or $\mathcal{D}$) [1].

$\mathcal{S}$ (Source)		$\mathcal{D}$ (Destination)
	Start of TLS Session	
Derive session key K	$\xleftrightarrow{\text{Mutual authentication, Key exchange}}$	Derive session key K
	Attestation of $\mathcal{S}$ and $\mathcal{D}$	
Pick $N_s \in \{0,1\}^n$	$\xrightarrow{\{N_s\}_K}$	
Verify N$_s$	$\xleftarrow{\{m_1\}_K}$	$m_1 = Sign_{AIK}(PCR \mid\mid N_s)$
	$\mathcal{S}$ verifies $\mathcal{D}$ platform integrity. A similar attestation of $\mathcal{S}$ is also performed by $\mathcal{D}$. If these verifications fail, the protocol is aborted.	
	VM-vTPM Transfer	
Suspend and lock the VM and its vTPM	$\xleftarrow{\{N_d\}_K}$	Pick $N_d \in \{0,1\}^n$
$m_2 = VM\mid\mid vTPM \mid\mid N_d$	$\xrightarrow{\{m_2\}_K}$	Verify N_d.
	$\xleftarrow{\{DELETE\}_K}$	Verify VM integrity
Delete VM and vTPM	$\xrightarrow{\{RESUME\}_K}$	Import VM-vTPM Resume VM and its vTPM
Delete K		Delete K
	End of TLS Session	

Figure 3: Sketch of a secure VM-vTPM migration protocol. The session key K is derived using key exchange protocols (e.g., TLS handshake) and used only for the current protocol instance. $\mathcal{S}$ and $\mathcal{D}$ verify each other's integrity by examining PCRs signed using an AIK key obtained from a Privacy CA. $Sign_X(Y)$ refers to the signature of Y using key X, n is a security parameter.

directly detected by $\mathcal{S}$. Since we assume that $\mathcal{A}$ cannot modify software on $\mathcal{S}$ and/or $\mathcal{D}$ during the migration process, little can be done by $\mathcal{A}$ to convince $\mathcal{S}$ that its machine is "honest", while it hosts in reality malicious/compromised software (Requirement 3). Given that the integrity of $\mathcal{S}$ is also verified, the authenticity of the migration initiation can also be ensured. This is the case since $\mathcal{D}$ verifies that it is interacting with an "honest" source and therefore can trust that $\mathcal{S}$ will abide by the protocol specification. This also includes trusting that $\mathcal{S}$ *(i)* will initiate the migration process upon the request of $\mathcal{P}$ (Requirement 2) and *(ii)* will *securely* delete the key K at the end of the migration process [22,23].

Note that this trust does not extend to the contents of the migrated VM. Indeed, while $\mathcal{S}$ might be "honest", the VM itself might be compromised by $\mathcal{A}$ prior to the start of the migration process. This use-case is countered by requiring that $\mathcal{D}$ checks the integrity of the migrated VM after the data transfer. As a result, $\mathcal{D}$ can ensure that only correct VMs can be migrated to its environment, thus conforming with Requirement 3 (Section 2).

4. PRACTICAL CONSIDERATIONS

So far, we have discussed secure VM-vTPM migration in the context of VM migration using the suspend- transfer-resume paradigm where the vTPM is a process inside the VM itself. Here, we discuss several challenges in extending it to alternative VM migration mechanisms (e.g., live migration) and vTPM architectures (e.g., each vTPM inside a separate VM, all vTPMs in a separate privileged VM). The performance of secure VM-vTPM migration is critical to its adoption and optimizing it requires understanding the nature of the overhead that it imposes. For this purpose, we present a prototype implementation and an initial study of the overhead in terms of end-to-end migration time and CPU usage.

4.1 Feasibility Study

In order to demonstrate the feasibility of our secure VM-vTPM migration protocol, we implemented a prototype and integrated it in the Xen hypervisor [9]. Our implementation emulates suspended VM-vTPM migration where each VM runs its own vTPM in the user space. We also show preliminary performance results on the overhead incurred by securing VM-vTPM migration in terms of end-to-end migration time and CPU usage. Finally, we discuss additional protocol characteristics and possible optimizations.

Implementation setup

In our implementation, we considered a private cloud virtual environment similar to the one described in Figure 1. We used two Thinkpad W510 (1.73 GHz, 8 GB RAM) machines with identical hardware as migration source ($\mathcal{S}$) and destination ($\mathcal{D}$). Both machines were configured to use the 64-bit Xen hypervisor (version 4.0.2-rc2-pre). The virtual machines (VM) running on $\mathcal{S}$ and $\mathcal{D}$ had their configuration files, disk and swap spaces on a separate NFS shared server. Each VM instance had its own virtual TPM running in user space. $\mathcal{S}$, $\mathcal{D}$ and the NFS server were on the same 1 GB Ethernet local network (LAN). Therefore, the migration protocol only transfers the VM-vTPM RAM im-

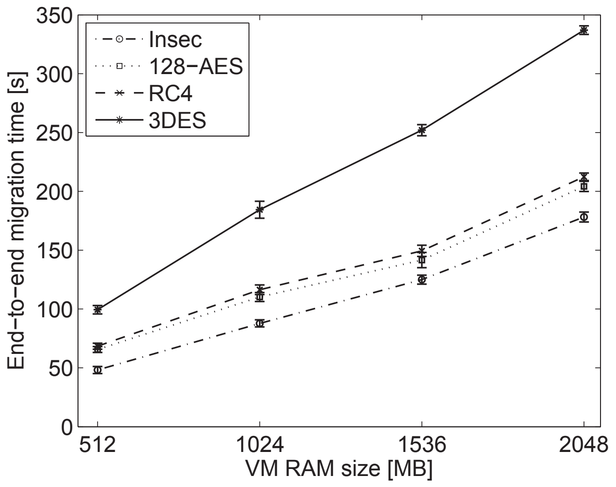


Figure 4: End-to-end VM-vTPM migration downtime for different encryption ciphers and VM RAM sizes. The results are validated over five independent migration protocol executions. We also show the corresponding 95% confidence intervals.

age. We note that this configuration is common in private cloud environments [24, 25].

Our implementation leverages on the Python-based Xen VM suspension and resumption. We used the TPM emulator [26] as a vTPM and executed it within the VM itself. We integrated OpenSSL (version 0.9.8o) to establish an authentic and secure channel and used the Privacy CA [27] to obtain AIKs for remote attestation. For simplicity, we only used boot time integrity measurements during attestation. We also implemented an insecure version of VM migration which simply transfers the VM-vTPM using the standard TCP socket interface. This implementation was used for comparison. Our source code is open-source and available for download at [10].

Preliminary results

One of the most common metrics used for evaluating the performance of migration protocols is the total end-to-end VM migration downtime [24, 25, 28]. It is defined as the time elapsed between the initiation of the migration protocol at the source and the completion of the VM migration at the destination. It is indicative of the actual VM downtime perceived by the end user. We measured the end-to-end migration time for different VM RAM sizes and encryption ciphers. Figure 4 shows the results validated with five independent migration protocol executions. For VM RAM sizes of 1 GB (typical in private cloud environments), the secure protocol with 128-AES encryption completed within approximately 110 s, that is 20% slower than the insecure version (88 s). We note that the absolute values can only significantly improve if the industry-standard virtualized hardware is used.

We further investigated the underlying overheads in the migration process by considering the CPU usage. The CPU time consumed by a process is indicative of the actual overhead imposed on the system as well as the contributions of the different components. We used the Google CPU profiler, part of the Google-perf tools [29] to instrument and measure the CPU usage during migration. Figures 5(a) and 5(b) show the distribution of CPU time between I/O and cryptographic operations measured for different VM RAM sizes and encryption ciphers. We notice that most of the time is spent in cryptographic (encryption, secure hashing) and optimized SSL I/O operations during the secure transfer of the VM RAM image. Note that this overhead is common to all secure solutions. We should also note that the overhead on Privacy CA and key regeneration in our design is independent of the number of migrated vTPM keys. In terms of the overhead due to attestation, this overhead depends on the type of properties being measured and the size of the measurement target. We note that the time required to measure hypervisor static properties is typically of the order of tens of milliseconds [19, 20].

The incurred security overhead may be tolerable in applications without strict timing constraints (e.g., email). In the case of time sensitive applications (e.g., video streaming), migration process optimizations would be required in order to reduce the total security overhead. These would include hardware and software as well as cryptographic-related optimizations.

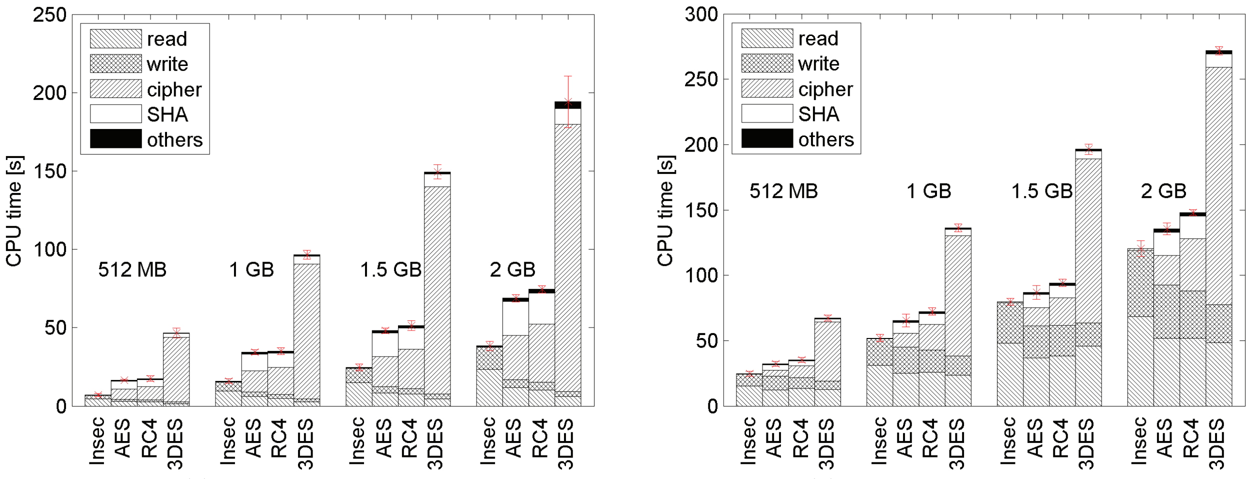
4.2 Discussion

In order to reduce the total migration downtime of time sensitive applications running on a VM, one direction is to consider performing live VM-vTPM migration [24, 25, 28]. However, secure live migration of VMs with vTPMs requires synchronizing VM-vTPM state during the transfer. Given that each vTPM (in our implementation) runs as a process within its own VM, existing memory synchronization techniques can be used to synchronize the VM-vTPM state on the source and destination during migration. This allows the resumption of the VM on the destination before the complete transfer of VM-vTPM state [24]. We note however that live migration of the vTPM may require specific VM memory partitioning in order to ensure that the vTPM state is transferred at once to avoid state corruption. The feasibility of such a secure VM-vTPM live migration approach needs more detailed investigation.

Other vTPM architecture designs could also help to improve performance. All vTPMs on a given hardware platform can run on separate, dedicated VM referred to as vTPM manager [4]. Such a dedicated VM can be migrated using secure VM migration as well. It is also easy to associate each vTPM to its VM in such cases. However, when all vTPMs run in a privileged VM, additional process migration techniques have to be devised to migrate the vTPMs. Further work is required to devise such migration and compare its properties and performance to the approach implemented in this work.

5. RELATED WORK

Previous work on VM-vTPM migration includes several protocol proposals in [4, 5, 7] and an implementation in the Xen hypervisor [9]. All protocols meet the confidentiality of the VM-vTPM during migration. However, only [4] used an integrity protection mechanism to ensure that migrated data has not been tampered with. For the other protocols, this is implicitly handled by the encryption mechanism in a sense that any modification of the encrypted VM image would re-



(a) Source CPU usage (b) Destination CPU usage

Figure 5: CPU usage of prototype implementation for different encryption ciphers and VM RAM sizes. The results are validated over five independent migration protocol executions. We also show the corresponding 95% confidence intervals.

sult with high likelihood in problems with VM resumption at the destination. There are no means for the migration process to understand the causes of failed resumption (e.g., due to intentionally corrupted VM image during transfer or server execution problems). Measuring the system integrity before migration was suggested in [5, 7]. While in [5], the integrity of the destination is checked as part of the secure channel establishment, in [7] the process is not completely specified. None of these works checked the integrity of the migration initiator (source). Furthermore, the initiator authenticity was also not considered as a requirement in the above works. We believe that this is an important feature for DoS and collocation attack prevention.

Previous work on vTPM architectures include proposals for vTPM key hierarchies [4–8]. Creating vEK credentials using the underlying TPM EK [5] does not comply with the EK usage restrictions. Moreover, using the TPM AIK to sign vEK or $vAIK$ credentials makes these keys non-migratable and hence, require fresh key generation at the destination. Finally, obtaining vEK or $vAIK$ credentials from a trusted third party (Privacy CA) increases the external dependencies and is not easy to realize in practice. Besides, one would have to inform the Privacy CA on every migration because the only basis for issuing such a credential can be the underlying platform. In [6], it is unclear how vTPMs can be migrated if the TPM EK and AIK are shared between several vTPMs. In [8], generation of vTPM keys and credentials has been only discussed from the perspective of the types of TPM keys that could be used in a virtual TPM.

Virtual machines (VMs) can be migrated in three ways, namely, stop-transfer-start paradigm, suspend-transfer-resume paradigm and using live migration in increasing order of efficiency [24]. The first two techniques either stop or suspend the VM before its transfer to the destination respectively. Live migration techniques are the most popular because they minimize VM downtime [24, 25]. A summary of live migration techniques can be found in [28]. Furthermore, VM migration may involve the transfer of the VM RAM only or the transfer of the VM RAM along with the disk image [25].

6. CONCLUDING REMARKS

In this work, we considered the problem of enabling secure VM-vTPM migration in private cloud virtualized environments. We analyzed the requirements for secure VM-vTPM migration in internal virtualized environments. We also proposed a vTPM key hierarchy that provides robust functionality to construct secure VM-vTPM migration protocols. Our key hierarchy is compliant with the TPM key usage recommendations, minimizes key regeneration after vTPM migration and prevents vTPM transaction linking. By leveraging on this hierarchy, we proposed and analyzed a secure VM-vTPM protocol and we evaluated its performance by means of implementation using the Xen hypervisor. This implementation demonstrates that our proposed secure VM-vTPM migration scheme can be directly integrated in open-source virtual systems. Preliminary results on the performance of our scheme showed that—compared to the cost of encrypting data—our migration scheme only incurs negligible overhead in the regeneration of the vTPM keys at the destination. The implementation is open-source and available online [10]. Finally, we also discussed how our scheme can be extended to support live migration of VM-vTPMs.

In terms of future work, we intend to extend the implementation to support different vTPM key hierarchies and analyze the performance of VM-vTPM migration in realistic virtualized computing environments. We also plan to extend our work to possibly secure live VM-vTPM migration.

Acknowledgements

The authors thank Dr. Aurélien Francillon for his help during the implementation. The authors would also like to acknowledge the anonymous reviewers for their helpful feedback and comments. The work presented in this paper was partly supported by the Swiss National Science Foundation under Grant 200021-127294.

7. REFERENCES

[1] TCG Architecture Overview, v1.4. http://www.trustedcomputinggroup.org.

[2] Amazon Elastic Compute Cloud, http://aws.amazon.com/ec2/.

[3] Tal Garfinkel and Mendel Rosenblum. When virtual is harder than real: security challenges in virtual machine based computing environments. In *HOTOS'05: Proceedings of the 10th conference on Hot Topics in Operating Systems*, pages 20–20, 2005.

[4] Stefan Berger, Ramón Cáceres, Kenneth A. Goldman, Ronald Perez, Reiner Sailer, and Leendert van Doorn. vTPM: Virtualizing the trusted platform module. In *USENIX-SS'06: Proceedings of the 15th conference on USENIX Security Symposium*, pages 305–320, 2006.

[5] Frederic Stumpf and Claudia Eckert. Enhancing Trusted Platform Modules with Hardware-Based Virtualization Techniques. In *SECURWARE '08: Proceedings of the 2008 Second International Conference on Emerging Security Information, Systems and Technologies*, pages 1–9, 2008.

[6] Paul England and Jork Loeser. Para-Virtualized TPM Sharing. In *Trust '08: Proceedings of the 1st international conference on Trusted Computing and Trust in Information Technologies*, pages 119–132, 2008.

[7] Ahmad-Reza Sadeghi, Christian Stüble, and Marcel Winandy. Property-Based TPM Virtualization. In *ISC '08: Proceedings of the 11th international conference on Information Security*, pages 1–16, 2008.

[8] Vincent Scarlata, Carlos Rozas, Monty Wiseman, David Grawrock, and Claire Vishik. TPM Virtualization: Building a General Framework. In *Trusted Computing*, pages 43–56, 2007.

[9] Xen hypervisor, http://www.xen.org.

[10] Secure VM-vTPM protocol implementation. http://www.syssec.ethz.ch/software/vtpm-migration.zip.

[11] Thomas Ristenpart, Eran Tromer, Hovav Shacham, and Stefan Savage. Hey, you, get off of my cloud: exploring information leakage in third-party compute clouds. In *CCS '09: Proceedings of the 16th ACM conference on Computer and communications security*, pages 199–212, 2009.

[12] Reiner Sailer, Xiaolan Zhang, Trent Jaeger, and Leendert van Doorn. Design and implementation of a TCG-based integrity measurement architecture. In *SSYM'04: Proceedings of the 13th conference on USENIX Security Symposium*, pages 16–16, 2004.

[13] Zhi Wang and Xuxian Jiang. Hypersafe: A lightweight approach to provide lifetime hypervisor control-flow integrity. In *Proceedings of the 2010 IEEE Symposium on Security and Privacy*, pages 380–395, 2010.

[14] Intel trusted execution technology. http://www.intel.com/Assets/en_US/PDF/whitepaper/323586.pdf.

[15] Jonathan M. McCune, Bryan J. Parno, Adrian Perrig, Michael K. Reiter, and Hiroshi Isozaki. Flicker: an execution infrastructure for tcb minimization. *SIGOPS Oper. Syst. Rev.*, 42:315–328, 2008.

[16] TPM Main Part 1 Design Principles. http://www.trustedcomputinggroup.org/resources.

[17] W. Diffie and M. Hellman. New directions in cryptography. *Information Theory, IEEE Transactions on*, 22:644 – 654, 1976.

[18] The transport layer security (TLS) protocol v 1.1. http://www.rfc-editor.org/rfc/pdfrfc/rfc4346.txt.pdf.

[19] Ahmed M. Azab, Peng Ning, Zhi Wang, Xuxian Jiang, Xiaolan Zhang, and Nathan C. Skalsky. Hypersentry: enabling stealthy in-context measurement of hypervisor integrity. In *Proceedings of the 17th ACM conference on Computer and communications security*, CCS '10, pages 38–49, 2010.

[20] Jiang Wang, Angelos Stavrou, and Anup Ghosh. Hypercheck: a hardware-assisted integrity monitor. In *Proceedings of the 13th international conference on Recent advances in intrusion detection*, RAID'10, pages 158–177, 2010.

[21] Yacine Gasmi, Ahmad-Reza Sadeghi, Patrick Stewin, Martin Unger, and N. Asokan. Beyond secure channels. In *STC '07: Proceedings of the 2007 ACM workshop on Scalable trusted computing*, pages 30–40, 2007.

[22] Radia Perlman. The ephemerizer: Making data disappear. *Journal of Information System Security*, 1:55 – 68, 2005.

[23] Radia Perlman. File system design with assured delete. In *In Proceedings of the Network and Distributed System Security Symposium (NDSS)*, 2007.

[24] Christopher Clark, Keir Fraser, Steven H, Jakob Gorm Hansen, Eric Jul, Christian Limpach, Ian Pratt, and Andrew Warfield. Live Migration of Virtual Machines. In *Proceedings of the 2nd ACM/USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, pages 273–286, 2005.

[25] Robert Bradford, Evangelos Kotsovinos, Anja Feldmann, and Harald Schiöberg. Live wide-area migration of virtual machines including local persistent state. In *VEE '07: Proceedings of the 3rd international conference on Virtual execution environments*, pages 169–179, 2007.

[26] Mario Strasser and Heiko Stamer. A Software-Based Trusted Platform Module Emulator. In *Trust '08: Proceedings of the 1st international conference on Trusted Computing and Trust in Information Technologies*, pages 33–47, 2008.

[27] Privacy certificate authority. http://www.privacyca.com.

[28] Michael R. Hines, Umesh Deshpande, and Kartik Gopalan. Post copy live migration of virtual machines citation. *ACM SIGOPS Operating Systems Review*, 43(3):14–26, 2009.

[29] Google Perftools, http://code.google.com/p/google-perftools/.

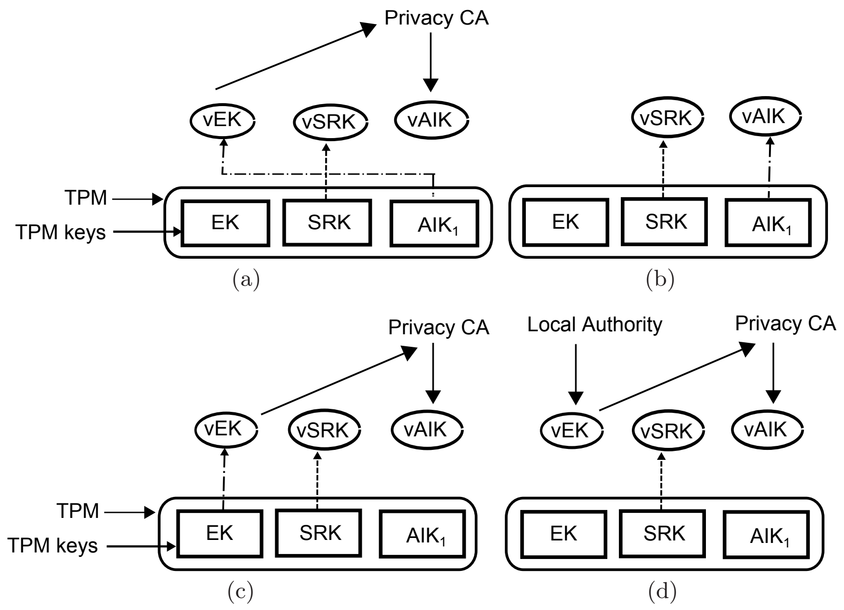


Figure 6: Existing vTPM key hierarchy designs. These designs range from tightly coupled TPM and vTPM keys to fully independent configurations which only depend on an external authority.

APPENDIX

A. PREVIOUS DESIGNS OF VTPM ARCHITECTURES

In [4], Berger et al. discuss the trade-offs between keeping vTPM key hierarchy independent from the TPM key hierarchy (as in Figure 6(d)) and linking it to the TPM via the AIK (as in Figures 6(a), 6(b)). The authors claim that the design in Figure 6(d) *(i)* minimizes the key and credential regeneration that are required after migration and *(ii)* complies with the procedure of obtaining AIKs from a Privacy CA. Sadeghi et al. [7] also adopt a similar design. In [5], Stumpf et al. propose a vTPM key hierarchy which uses the TPM EK to sign the vEK (see Figure 6(c)). Here, the AIK is obtained from a Privacy CA as in the case of the TPM.

Exposing Invisible Timing-based Traffic Watermarks with BACKLIT

Xiapu Luo[‡], Peng Zhou[‡], Junjie Zhang[§],
Roberto Perdisci[†], Wenke Lee[§], and Rocky K. C. Chang[‡]
The Hong Kong Polytechnic University[‡] Georgia Institute of Technology[§] University of Georgia[†]
{csxluo|cspzhouroc|csrchang}@comp.polyu.edu.hk,
{jjzhang|wenke}@cc.gatech.edu, perdisci@cs.uga.edu

ABSTRACT

Traffic watermarking is an important element in many network security and privacy applications, such as tracing botnet C&C communications and deanonymizing peer-to-peer VoIP calls. The state-of-the-art traffic watermarking schemes are usually based on packet timing information and they are notoriously difficult to detect. In this paper, we show *for the first time* that even the most sophisticated timing-based watermarking schemes (e.g., RAINBOW and SWIRL) are not invisible by proposing a new detection system called BACKLIT. BACKLIT is designed according to the observation that any practical timing-based traffic watermark will cause noticeable alterations in the intrinsic timing features typical of TCP flows. We propose five metrics that are sufficient for detecting four state-of-the-art traffic watermarks for bulk transfer and interactive traffic. BACKLIT can be easily deployed in stepping stones and anonymity networks (e.g., Tor), because it does not rely on strong assumptions and can be realized in an active or passive mode. We have conducted extensive experiments to evaluate BACKLIT's detection performance using the PlanetLab platform. The results show that BACKLIT can detect watermarked network flows with high accuracy and few false positives.

1. INTRODUCTION

A traffic watermark is a piece of information embedded into a network flow. If the traffic watermark is retained in the network flow, it can be used to correlate network traffic observed at different network locations. Therefore, traffic watermarking has many important network security and privacy applications, including identifying stepping stones used by attackers to hide their true physical locations [1,2], tracking users visiting sensitive (e.g., pro-terrorism) Web sites [3], tracing communications among bot-compromised machines [4], and tracking anonymous peer-to-peer VoIP calls [5].

The state-of-the-art traffic watermarking schemes embed *timing-based watermarks* into a network flow by artificially manipulating packet timing information, such as inflating and deflating inter-packet delay [2, 3, 6, 7]. The timing-based watermarks have seen continuous improvement over the last few years, evolving from an initial sole focus on usability [1] to robustness [2, 3] and invisibility [6, 7]. The most recently proposed timing-based watermarks, such as RAINBOW [6] and SWIRL [7], are invisible as they can evade the existing detection schemes.

There are two general strategies for mitigating the traffic watermarking threat. The first strategy blindly removes the timing information brought by traffic watermarks through shaping and/or padding every flow no matter it is watermarked or not [8–10]. Although it is effective, this strategy introduces significant overhead such as high latency to all flows and useless packets consuming bandwidth, making them very difficult to scale. Another strategy is to first detect watermarked flows and then clean only the suspicious flows. Compared to the first strategy, the detection strategy introduces much less overhead for identifying watermarked flows and has the advantages of handling only the suspicious flows and discovering the existence of watermark stamping devices.

Detecting timing-based watermarks is a very challenging problem, because they do not have a fixed signature. So far, only a few detection mechanisms have been proposed [11, 12]. However, they suffer from two main drawbacks. First, they fail to detect advanced traffic watermarking schemes, such as RAINBOW and SWIRL, because these watermarking schemes may not cause anomalies in the features used by those detections mechanisms. Second, they rely on strong assumptions for the detection. For example, the detection system PNR (which is named by concatenating the first letters of the three authors' last names) [11] assumes that the traffic sender can inject a timestamp into each packet, but this is not feasible in many scenarios (e.g., a public Web server not controlled by the detection system). PNR's another assumption that one-way packet delay follows the Gaussian distribution also does not always hold [13]. In Multi-Flow Attack (MFA) [12], the assumptions of watermarking multiple flows with the same traffic watermark and modeling the flows as a Markov modulated Poisson process (MMPP) do not always hold either [14]. Kiyavash et al. [15] later showed how MFA can be easily evaded by randomizing the locations of watermarks or using different watermarks for different network flows.

In this paper, we demonstrate *for the first time* that even the most advanced timing-based traffic watermarking scheme can be detected by a practical system. We show this by proposing BACKLIT, a new system capable of detecting four advanced watermarking schemes, including the "invisible" RAINBOW and SWIRL, the interval based watermarking (IBW) scheme, and the interval centroid based watermarking (ICBW) scheme. We design BACKLIT based on the observation that any practical watermark that artificially perturbs packet timing information will cause noticeable alterations in the intrinsic *timing features* typical of TCP flows. We first select the TCP features that can be used for detection purposes, then design metrics to quantify these features, and finally employ the one-class

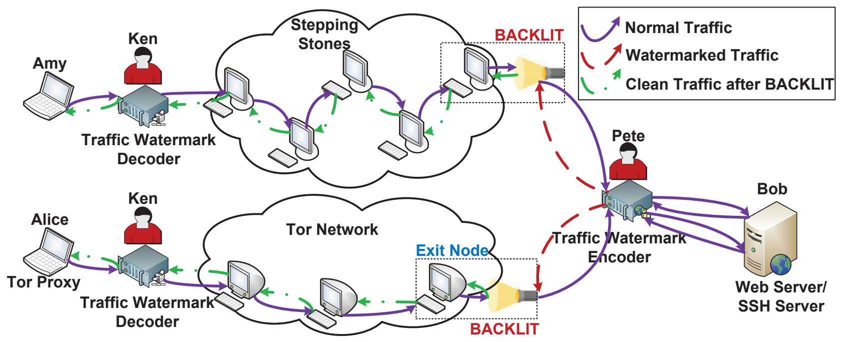


Figure 1: Typical scenarios of using traffic watermarks and BACKLIT.

classifier to detect watermarked flows based on the set of metrics. Moreover, unlike previous studies that use synthetic and replayed traffic for their evaluation [2,6,7,12], we evaluate BACKLIT using live HTTP traffic and SSH traffic through twelve PlanetLab nodes deployed around the world. The evaluation shows that BACKLIT can detect the four timing-based watermarking schemes with high accuracy, often achieving a 100% detection rate and low false positive rate.

Besides the high detection performance, BACKLIT can also be readily deployed in real network environments. BACKLIT is much more practically feasible than PNR and MFA, because it does not rely on strong assumptions required for PNR and MFA. BACKLIT is more flexible than PNR and MFA, because BACKLIT can be deployed as an active or passive system by either generating real requests or relaying requests from other users to induce response packets for detection. MFA is a passive-only system that makes a decision from observed traffic, and PNR is an active-only system that sends customized packets to detect the existence of traffic watermarks. Moreover, unlike PNR, BACKLIT does not need to modify packets and install special software at the remote server. In this paper, we focus on TCP flows, because TCP carries about 90% of Internet traffic [16], and popular anonymity networks (e.g., Tor [17]) currently only support TCP-based applications. However, the same idea could be applied to other end-to-end protocols.

The remainder of the paper is organized as follows. Section 2 describes the threat model and Section 3 introduces the state-of-the-art timing-based traffic watermarks. The design and implementation of BACKLIT are detailed in Section 4 and Section 5, respectively. We present evaluation results in Section 6. After introducing related work in Section 7, we conclude the paper with future work in Section 8.

2. THREAT MODEL

Figure 1 illustrates typical scenarios of using traffic watermarks. Bob runs a server that hosts *sensitive* content (e.g., WikiLeaks) that Alice and Amy visit through an anonymity network (e.g., Tor [17]) and a series of stepping stones [18], respectively. Pete, who has access to Bob's network traffic, embeds a sequence of traffic watermarks into the traffic originating from Bob in an attempt to trace back who is visiting Bob's server. Ken, Pete's associate, monitors the traffic directed towards Alice and Amy. If Ken can correctly decode the sequence of watermarks encoded by Pete, he can establish that Alice and Amy are visiting Bob's server.

The owner of the anonymity network or stepping stones can install BACKLIT to determine whether Bob's traffic is being watermarked. If that is the case, she can employ various packet padding and buffering strategies to remove traffic watermarks without affecting all incoming flows [8, 9]. BACKLIT either sends real requests (i.e., active mode) or employs relayed requests from other users (i.e., passive mode) to induce packets from Bob's server for detection. Therefore, Pete cannot distinguish whether the host running BACKLIT is actually relaying requests on behalf of a real user (e.g., Alice) or is attemping to discover Pete's presence. Thus, Pete cannot selectively avoid watermarking traffic directed from Bob to BACKLIT. We assume that Pete does not collude with Bob.

As shown in Figure 1, BACKLIT can be strategically deployed at an exit node of the Tor network or a stepping stone connecting directly to Bob's server to facilitate the detection. The deployment flexibility provides the advantage that the traffic observed by BACKLIT is less affected by noise in comparison with the traffic monitored by Ken, because the traffic observed by Ken has to pass through *more* hops within the anonymity network or stepping stones. Such deployment flexibility has also been used in PNR's [11] and MFA's [12] experiments where there is no relay node between the traffic watermark encoder and the detection system.

3. STATE-OF-THE-ART TIMING-BASED TRAFFIC WATERMARKS

This paper considers four state-of-the-art traffic watermarking schemes listed in Table 1. The interval based watermarking (IBW) scheme [2] and the interval centroid based watermarking (ICBW) scheme [3] do not consider invisibility in their design, but the more recently proposed RAINBOW and SWIRL do. In particular, RAINBOW and SWIRL introduce much shorter delay to the packets than IBW and ICBW. Although MFA can detect IBW and ICBW [12], they can evade the detection by stamping different watermarks to network flows and randomizing the location of watermarks [15]. As shown in subsequent sections, BACKLIT can detect all of these watermarking schemes. PNR [11] is not listed in Table 1 because there are no experiment results showing whether PNR can detect these traffic watermarks.

Table 1: The timing-based traffic watermarking schemes considered in this paper.

Traffic watermarking scheme	Designed for invisibility?	Inserted delay	Evades MFA?	Evades BACKLIT?
IBW [2]	No	Long	No	No
ICBW [3]	No	Long	No	No
SWIRL [7]	Yes	Short	Yes	No
RAINBOW [6]	Yes	Short	Yes	No

3.1 IBW

IBW adjusts the number of packets in selected time intervals to embed watermarks [2]. Starting from a random offset, the flow to be watermarked is divided into a series of intervals I_i of the same length T_{IBW}. Pairs of consecutive intervals are randomly chosen to encode a bit. As shown in Figures 2(a) and 2(b), to embed a 0, IBW forces the number of packets in interval I_k to be larger than that in interval I_{k+1} by delaying all packets within I_{k-1} to I_k and those within I_{k+1} to I_{k+2}. To embed an 1, IBW delays packets in I_k to I_{k+1}, so that I_k has fewer packets than I_{k+1}.

3.2 ICBW

ICBW varies the centroid of the packets within selected time intervals to encode watermarks [3]. Starting from a random offset, the target flow is partitioned into a series of intervals of the same length T_{ICBW}. As shown in Figure 2(c), to alter the centroid of the packets in a given interval, ICBW changes each packet's relative time to the start of its interval from ΔT to $\Delta T^{'} = \alpha_{ICBW} + \frac{(T_{ICBW} - \alpha_{ICBW})\Delta T}{T_{ICBW}}$, where α_{ICBW} determines the maximum delay applied to the packets. Before encoding a bit, ICBW randomly selects two sets of intervals, say I_A and I_B, from a series of consecutive intervals. To encode a 0 (or 1), ICBW forces the centroid of the packets in I_B (or I_A) to be larger than that in I_A (or I_B) by delaying packets in I_B (or I_A).

3.3 SWIRL

SWIRL changes the locations of packets within selected time slots to encode watermarks [7]. Starting from a random offset, SWIRL divides a flow into a series of intervals of the same length T_{SWIRL}. It further defines two intervals: the mark interval and the basic interval. As shown in Figure 2(d), SWIRL partitions a mark interval T_{SWIRL} into r subintervals of length T_{SWIRL}/r and then splits each subinterval into m slots. To embed traffic watermarks, SWIRL first selects a slot i in each subinterval j according to a permutation $\pi^{(j)}(s) = i$, where s is a variable determined by the basic interval's centroid, and then delays each packet from the original slot to the selected slot. If the original slot's index is larger than that of the selected slot, packets will be postponed to the selected slot in the next subinterval. The delay added to each packet is in the range of $[0, \frac{2(m-1)T_{SWIRL}}{mr}]$ [7].

3.4 RAINBOW

RAINBOW inflates or deflates an inter-packet delay (IPD) by T_{RB} with equal probability [6]. Let T_{RB}^{i} $(i = 1, \ldots)$ denote the adjustment made to the ith IPD. If $T_{RB}^{i} > 0$, the IPD will be increased; otherwise, it will be shortened. Since a packet cannot be delayed for a negative amount of time, RAINBOW sets the delay of the jth $(j = 1, \ldots)$ packet to $T_j = T_0 + \sum_{i=1}^{j-1} T_{RB}^{i}$, where T_0 is a parameter large enough to ensure $T_j \geq 0$. Figure 2(e) gives an example of RAINBOW.

4. BACKLIT

Table 2 summarizes the differences among PNR [11], MFA [12], and BACKLIT. BACKLIT is more practically feasible, because unlike PNR and MFA, BACKLIT neither needs a timestamp in each packet and the cooperation of the remote server, nor relies on traffic distribution assumptions. Instead, BACKLIT exploits TCP's intrinsic features to expose various traffic watermarks, which are elaborated in Section 4.1.

4.1 Feature selection

Since all timing-based traffic watermarking schemes delay packets, BACKLIT leverages three TCP features that will be noticeably altered by traffic watermarks for detection: (1) *request-response time* (RRT), the time elapsed between sending the last packet of a request message from BACKLIT and receiving the first response packet from the remote server, (2) *inter-packet delay* (IPD), the time between two consecutive packets from the remote server, and (3) *burst size*, the number of TCP packets sent back-to-back [19]. We calculate the burst size using the method in [19].

Figure 3(a) shows an example of *normal* IPD, RRT, and burst size in the absence of traffic watermarks. Upon receiving a request packet, the server sends back a burst of four packets. If a traffic watermarking scheme delays the fourth response packet, a disturbed IPD (i.e., denoted by IPD') will be observed, as shown in Figure 3(b). As a side effect of inflating the IPDs, the burst size will also be changed. Particularly, if the delay induced by a watermark is large enough, say not less than the round-trip time (RTT), we can discern two bursts of packets: one has three packets and the other has one. Similarly, if a traffic watermarking scheme delays the first response packet (e.g., the second delay in Figure 3(b)), we could observe an increased RRT (i.e., denoted by RRT').

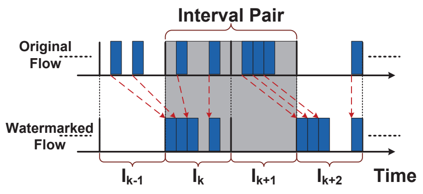
We roughly divide the network traffic into two types: (1) bulk transfer traffic that contains much more response packets than request packets. For example, an HTTP request that initiates a large file download will cause bulk transfer traffic. (2) interactive traffic that has similar number of request packets and response packets. For instance, an SSH session where the client sends a number of shell commands and receives relatively short responses (in terms of content) from the server will result in interactive traffic. Although, to our best knowledge, the types of traffic to which the existing traffic watermarking schemes can be applied are not explicitly documented, we assume that they can be employed to watermark both bulk transfer traffic and interactive traffic.

We discuss the selected features in bulk transfer traffic and interactive traffic individually for the ease of explaining the anomalies caused by different traffic watermarks. In bulk transfer traffic, we generally observe a large number of IPD samples and bursts of packets. Hence, BACKLIT employs them to perform the detection. In interactive traffic, we generally obtain many RRT samples triggered by requests. If the response to a request does not have many packets, the number of IPD samples may be limited and the burst size will not be determined by the TCP congestion control mechanism. In this case, BACKLIT relies only on RRT samples to carry out the detection. Since a network flow can be either bulk transfer traffic or interactive traffic or mixed traffic, BACKLIT selects suitable features for detection. For example, if an HTTP client sends many requests, each of which induces a short response, BACKLIT can obtain many RRT samples and then use them to carry out the detection. As another example, issuing a command like `ls -R /`, which asks a server to respond with a list of all files and directories on disk, will trigger many back-to-back packets from the server. In this case, BACKLIT can exploit the IPD samples and the burst sizes to perform the detection.

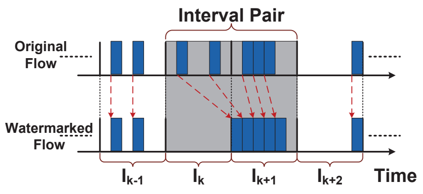
In the next three subsections, we use data from watermarked HTTP flows for bulk transfer traffic and watermarked SSH flows for interactive traffic to elaborate on the anomalies caused by different traffic watermarks on selected features. We adopt the traffic watermarking parameters given in the first row of Table 6 in Section 6. The experiments were conducted between two hosts with a minimum RTT of 0.184 seconds if there is no other specification.

4.1.1 RRT

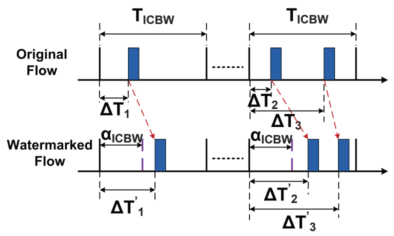
RAINBOW and SWIRL cause two types of anomalies in RRTs.



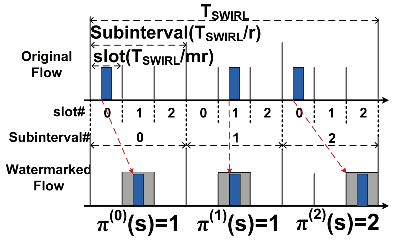
(a) Example of IBW (Encoding 0).



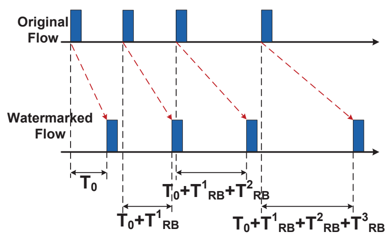
(b) Example of IBW (Encoding 1).



(c) Example of ICBW.



(d) Example of SWIRL.



(e) Example of RAINBOW.

Figure 2: Examples of IBW, ICBW, SWIRL and RAINBOW.

Table 2: Comparison of PNR, MFA, and BACKLIT.

Detection method	Timestamp required?	Assumes an ideal distribution?	Features	Mode	Requires cooperation from remote server ?
PNR [11]	Yes	Yes (Gaussian)	One-way packet delay	Active	Yes
MFA [13]	No	Yes (MMPP)	IPD	Passive	No
BACKLIT	No	No	IPD, RRT, burst size	Hybrid	No

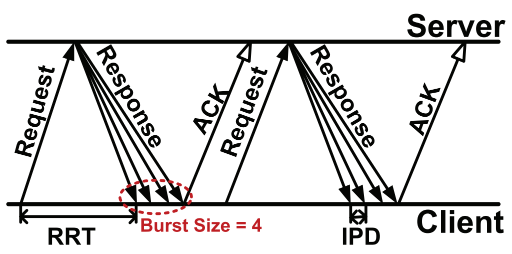
First, since most response packets will be delayed for some periods, the mean and the variance of disturbed RRTs will be larger than those of normal RRTs. Second, the absolute difference of consecutive RRTs, denoted as $|\Delta\text{RRT}|$, is equal to one or multiple T_{RB}s in traffic watermarked by RAINBOW and zero or multiple $\frac{T_{SWIRL}}{mr}$ in traffic watermarked by SWIRL if there is no noise. It is worth noting that the case of $|\Delta\text{RRT}| = 0$ in traffic watermarked by SWIRL appears only when consecutive responses are delayed for the same period. The probability of observing such case is extremely small because each response packet is delayed for a random period and the time when SWIRL observes a response is determined by the sending time of a request and the processing time of the server, both of which are random to SWIRL. BACKLIT leverages the second observation to detect RAINBOW and SWIRL because $|\Delta\text{RRT}|$ could be used to estimate the parameters employed in RAINBOW and SWIRL. Figure 4(a) illustrates the distribution of ΔRRT in normal SSH traffic and that in traffic watermarked by RAINBOW and SWIRL. The bin width is 0.005 seconds. These two distributions are quite different as most ΔRRTs in normal traffic are in the range of $[-0.005, 0.005]$, whereas the majority of ΔRRTs in watermarked traffic fall in the range of $[-0.05, 0.05]$.

IBW and ICBW will inflate RRTs. To clean packets in an interval, IBW delays them to the next interval. Depending on their locations in an interval, response packets will be delayed for a period in $(0, T_{IBW}]$. Similarly, ICBW delays response packets in selected intervals for a period in $(0, \alpha_{ICBW}]$. Figure 4(b) shows the distributions of RRTs from normal SSH traffic, traffic watermarked by IBW, and traffic watermarked by ICBW. It is clear that IBW and ICBW cause a set of abnormal RRTs that are larger than the normal RRTs.

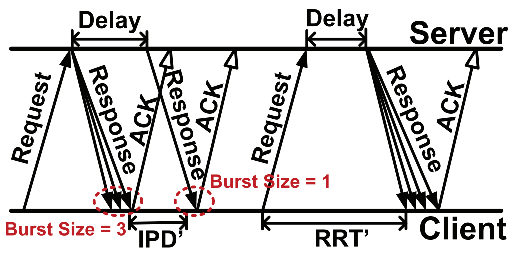
4.1.2 IPD

All timing-based watermarking schemes change IPDs. Let ξ_{RTT} be the minimum RTT between BACKLIT and a remote server. BACKLIT employs IPDs larger than ξ_{RTT} to detect IBW and ICBW, because they introduce large delays to packets. In normal bulk transfer traffic, such large IPD samples are usually found from the intervals between two bursts of packets and such observation has been used to estimate a TCP flow's RTT [20]. When IBW and ICBW delay a batch of packets, abnormal IPDs will be observed between the first packet in the current burst and the last packet in the previous burst. Figure 4(c) illustrates IPDs that are larger than ξ_{RTT} in normal HTTP traffic and compares them with IPDs that are larger than ξ_{RTT} in traffic watermarked by ICBW and IBW. We can see that in normal traffic such IPDs are close to ξ_{RTT}, whereas in traffic watermarked by IBW and ICBW the IPDs are in the range of $(0.5, 0.7)$ seconds and $(0.35, 0.54)$ seconds, respectively. Note that the IPDs close to the lower bound (i.e., 0.5 and 0.35) are due to the delaying of packets within a burst, and those close to the upper bound (i.e., 0.7 and 0.54) originate from the delaying of the first packet in a burst.

BACKLIT uses IPDs lower than ξ_{RTT} to detect RAINBOW and SWIRL, because they introduce smaller delays. In normal traffic, such IPD samples usually come from packets sent back-to-back. It is worth noting that if two packets are sent back-to-back, then the normal IPD is close to ξ_{CAP}, the time required by the bottleneck on the network path to transmit a packet [21]. This observation has been widely used to estimate bottleneck capacity [21]. ξ_{CAP} is generally very small, for example, a 10Mbps link needs only 1.2 ms to transmit a 1500-byte packet. Since RAINBOW and SWIRL



(a) Normal IPD, RRT, and burst size.



(b) Disturbed IPD (IPD'), RRT (RRT'), and burst size.

Figure 3: Normal and disturbed inter-packet delay, request-response time, and burst size.

do not know the path from the encoder to the decoder in advance (i.e., the path is what they want to trace), they cannot set T_{RB} and $\frac{T_{SWIRL}}{mr}$ according to ξ_{CAP}. Moreover, if two packets are not sent back-to-back, say the interval between these two packets being larger than ξ_{CAP}, then the IPD will be equal to the original interval if there is no cross traffic [21]. Therefore, although RAINBOW and SWIRL introduce a small delay to the packets, they disturb the distribution of IPDs less than ξ_{RTT}. Figure 4(d) plots the distribution of such IPDs in normal HTTP traffic along with that in traffic watermarked by RAINBOW and SWIRL. In normal traffic most IPDs approximate ξ_{CAP}, whereas in the traffic watermarked by RAINBOW or SWIRL the majority of IPDs scatter to 0.01-0.06 seconds.

Though RAINBOW may let T_{RB} be larger than or equal to RTT, BACKLIT can still detect it, because in the former case it causes similar anomalies as IBW and ICBW do, and in the latter case RAINBOW leads to abnormal burst size, as will be discussed next.

4.1.3 *Burst size*

Although RAINBOW may decrease the number of disturbed IPDs that are lower than ξ_{RTT} by setting T_{RB} to ξ_{RTT}, doing so will cause suspicious burst sizes. In bulk transfer traffic, the size of a burst is approximately equal to the TCP sender's congestion window. Since RAINBOW divides the packets in each burst into several small groups, many one-packet bursts can be observed. If there is enough data to send in bulk transfer traffic, a TCP sender seldom dispatches one packet in an RTT, except for the case in which the sender is in the timeout state (i.e., retransmitting lost packets) or in the fast retransmit state with a very small congestion window. We propose methods to filter out biased samples due to packet losses in Section 4.3.

To facilitate the explanation, we assume that there are only two kinds of IPDs: one is equal to ξ_{RTT}, and the other is equal to ξ_{CAP}. Since a one-packet burst occurs if two consecutive IPDs are equal to ξ_{RTT} according to the algorithm for computing burst size in [19], the probability of observing a one-packet burst is equal to $1 - P_{\mathfrak{w}}$ where $P_{\mathfrak{w}}$ is the probability that no consecutive IPDs are equal to ξ_{RTT}. Since RAINBOW will increase IPDs to ξ_{RTT} with probability 0.5, according to the Lemma 1 in the Appendix, we can get the probability of observing one-packet bursts as $1 - \frac{L}{2}/\binom{L}{\frac{L}{2}}$ from a sequence of L IPDs. This probability increases quickly with the length of L. In other words, when RAINBOW uses $T_{RB} = \xi_{RTT}$ to adjust the intervals between packets, we can observe many abnormal bursts that have only one packet.

Figure 4(e) shows the rate of one-packet bursts in normal HTTP flows and that in flows watermarked by RAINBOW with $T_{RB} \approx \xi_{RTT}$. The experiments were carried out between a host in Hong Kong and three PlanetLab nodes in Japan. It is obvious that the rate of one-packet bursts in normal flows is very small, whereas RAINBOW significantly increases it by letting $T_{RB} \approx \xi_{RTT}$.

4.2 Detection metrics and algorithm

BACKLIT adopts the anomaly detection approach and it marks a flow as a watermarked one if anomalies are found in any metric listed in Table 3. We elaborate on the metrics and the detection algorithm in Sections 4.2.1 and 4.2.2, respectively.

4.2.1 *Metrics*

Of the five metrics in Table 3, BACKLIT uses $\mathbb{R}$ and $\mathbb{T}$ to detect IBW and ICBW, and employs $\mathbb{Q}$, $\mathbb{D}$ and $\mathbb{Z}$ to discover RAINBOW and SWIRL. Let $\mathcal{F}$ represent a flow under inspection. We first describe how to calculate $\mathbb{R}$ and $\mathbb{T}$, because their computation procedures are similar, and then present how to compute $\mathbb{Q}$ and $\mathbb{D}$ for the same reason, and subsequently explain the calculation of $\mathbb{Z}$.

Let L denote a set of IPDs that are larger than ξ_{RTT} and $||L||$ be the number of such IPDs. We define a threshold $TH_{IPD}(k) = \mu_{IPD} + k\sigma_{IPD}$, where μ_{IPD} and σ^2_{IPD} are the mean and the variance of L in normal flows, respectively. Let $\mathbb{M}_{IPD}(k)$ denote the number of IPDs that are larger than $TH_{IPD}(k)$ in $\mathcal{F}$. We define $\mathbb{R}$ as the accumulative rate of IPDs larger than TH_{IPD}:

$$\mathbb{R} = \sum_{k=3}^{\varpi} \frac{\mathbb{M}_{IPD}(k)}{||L||}, \quad (1)$$

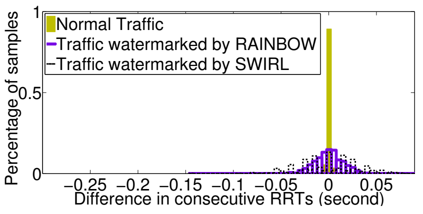
where ϖ is the minimum k that causes $\mathbb{M}_{IPD}(k) = 0$. More precisely, when k increases, the threshold TH_{IPD} becomes larger and $\mathbb{M}_{IPD}$ decreases. Since IPD is a limited value, there exists a k that leads to $\mathbb{M}_{IPD}(k) = 0$, assuming that $\sigma_{IPD} > 0$. We let k start from 3, because, according to the one-sided Chebyshev inequality [22], the probability of having a sample larger than such threshold is not greater than $\frac{1}{1+k^2} = 0.1$.

Similarly, let S represent a set of RRTs that are larger than ξ_{RTT} and $||S||$ be the number of such RRTs. We define a threshold $TH_{RRT}(k) = \mu_{RRT} + k\sigma_{RRT}$, where μ_{RRT} and σ^2_{RRT} are the mean and the variance of S in normal flows. Let $\mathbb{M}_{RRT}(k)$ be the number of RRTs that are larger than $TH_{RRT}(k)$ in $\mathcal{F}$. We define $\mathbb{T}$ as the accumulative rate of RRTs larger than TH_{RRT}:

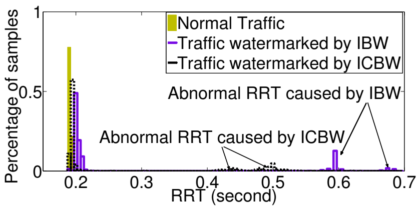
$$\mathbb{T} = \sum_{k=3}^{\omega} \frac{\mathbb{M}_{RRT}(k)}{||S||}, \quad (2)$$

where ω is the minimum k that causes $\mathbb{M}_{RRT}(k) = 0$.

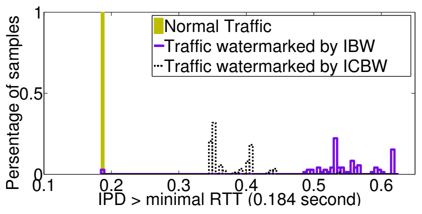
$\mathbb{Q}$ refers to the similarity between the distribution of IPDs that are lower than ξ_{RTT} in $\mathcal{F}$ and that in normal flows. $\mathbb{D}$ represents the similarity between the distribution of absolute ΔRRT (i.e., $|\Delta$RRT$|$) in $\mathcal{F}$ and that in normal flows. BACKLIT uses $\mathbb{Q}$ and $\mathbb{D}$ to detect RAINBOW and SWIRL, because they cause significant changes in the distribution of IPDs that are smaller than ξ_{RTT} and in the distribution of $|\Delta$RRT$|$.



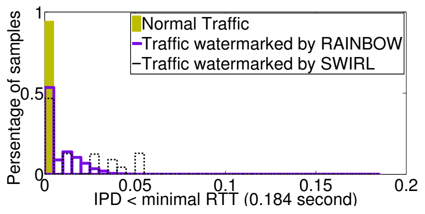
(a) Differences in consecutive RRTs.



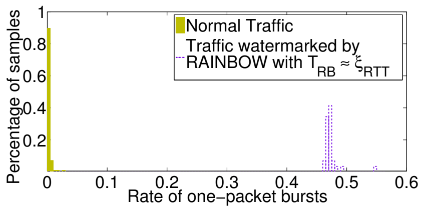
(b) RRTs.



(c) IPDs that are larger than minimal RTT.



(d) IPDs that are smaller than minimal RTT.



(e) Rate of one-packet bursts.

Figure 4: Features in normal traffic and watermarked traffic.

Table 3: Metrics used by BACKLIT to detect timing-based traffic watermarks.

Feature	Bulk transfer traffic (e.g., HTTP)				Interactive traffic (e.g., SSH)				Notation
	IBW	ICBW	RAINBOW	SWIRL	IBW	ICBW	RAINBOW	SWIRL	
Accumulative rate of IPDs larger than TH_{IPD}	$\surd$	$\surd$							$\mathbb{R}$
Accumulative rate of RRTs larger than TH_{RRT}					$\surd$	$\surd$			$\mathbb{T}$
Distribution similarity of IPDs smaller than ξ_{RTT}			$\surd$	$\surd$					$\mathbb{Q}$
Distribution similarity of $\|\Delta$RRT$\|$s							$\surd$	$\surd$	$\mathbb{D}$
Rate of one-packet bursts			$\surd$ if $T_{RB} \approx \xi_{RTT}$						$\mathbb{Z}$

For $\mathbb{Q}$ and $\mathbb{D}$, we calculate the difference between histograms to quantify the similarity between the distribution in normal traffic and that in a flow to be inspected. $\mathbb{D}$ is used as an example to illustrate the computation of this similarity. The method comprises two steps:

1. Construct a histogram, denoted as $H_{\Delta RTT}$, for a sequence of $|\Delta\text{RRT}|$s. Let ΔRRT_{max} be the maximum value of $|\Delta\text{RRT}|$. Divide the range $[0, \Delta\text{RRT}_{max}]$ into M disjoint subregions of equal size, called *histogram bins*. Let $H_{\Delta RTT}(i)$ ($i = 1, \ldots, M$) be the percentage of $|\Delta\text{RRT}|$s that fall into the ith histogram bin.
2. Compute the similarity between $H^{Normal}_{\Delta RTT}$ from normal flows and $H^{\mathcal{F}}_{\Delta RTT}$ from $\mathcal{F}$ as

$$\mathbb{D} = \sum_{i=1}^{\max\{M^{Normal}, M^{\mathcal{F}}\}} (H^{\mathcal{F}}_{\Delta RTT}(i) - H^{Normal}_{\Delta RTT}(i))^2, \quad (3)$$

where M^{Normal} and $M^{\mathcal{F}}$ are the number of bins in $H^{Normal}_{\Delta RTT}$ and $H^{\mathcal{F}}_{\Delta RTT}$, respectively. We set $H^{Normal}_{\Delta RTT}(j) = 0$ (or $H^{\mathcal{F}}_{\Delta RTT}(j) = 0$) if $j > M^{Normal}$ (or $j > M^{\mathcal{F}}$).

$\mathbb{Z}$ represents the ratio of the number of one-packet bursts to the total number of bursts. It is applied to bulk transfer traffic, because as explained in Section 4.1.3 RAINBOW will cause many one-packet bursts in bulk transfer traffic when $T_{RB} = \xi_{RTT}$.

Figures 5(a), 5(b), 5(c), and 5(d) show the distributions of $\mathbb{R}$ and $\mathbb{Q}$ in normal HTTP flows and those in HTTP flows watermarked by IBW, ICBW, RAINBOW and SWIRL using different parameter settings. Figures 6(a), 6(b), 6(c), and 6(d) show the CDFs of $\mathbb{T}$ and $\mathbb{D}$ obtained from normal SSH flows and those in watermarked SSH flows using different parameter settings. It can be seen that watermarked flows cause significant changes to $\mathbb{Q}$, $\mathbb{R}$, $\mathbb{T}$, and $\mathbb{D}$, indicating that these metrics can be used to effectively detect watermarked flows.

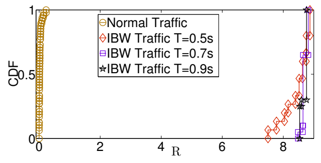
4.2.2 *Detection algorithm*

BACKLIT employs the one-class classifier [23] to capture anomalies in the five metrics, because it does not have watermarked flows when training the classifier. The one-class (OC) classifier learns from a single class of samples, which is labeled as the *target class*. The OC classifier identifies a boundary around the available data from the target class such that it includes as much data from the target class as possible, at the same time minimizing the chance of accepting *outliers* [23]. In other words, each datum is classified as a member of either the target class or the *outlier class*. The boundary is determined during the training with a given pre-reject ratio (P_R) which is the fraction of rejected training data for the sake of excluding possible noise in the training data [23]. If all the training samples belong to the target class, P_R can be regarded as a tolerable false positive rate [24].

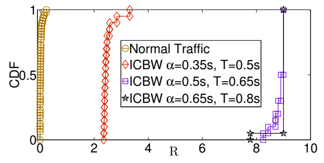
When using the one-class classifier to detect traffic watermarks, the target (outlier) class corresponds to the class of non-watermarked (watermarked) network flows. We use the false positive rate and the detection rate to evaluate BACKLIT. The former is ratio of the number of non-watermarked flows that are mistakenly regarded as watermarked flows to the total number of non-watermarked flows. The latter is the ratio of the number of watermarked flows that are detected by BACKLIT to the total number of watermarked flows.

4.3 Normal profile creation

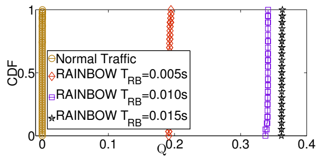
Before carrying out the detection, we collect data for constructing the normal profiles of RRT, IPD, and burst size before the de-



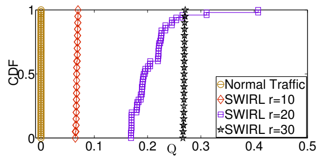
(a) Detect IBW through $\mathbb{R}$.



(b) Detect ICBW through $\mathbb{R}$.

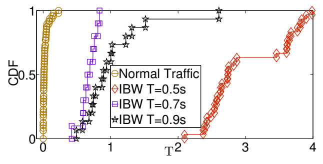


(c) Detect RAINBOW through $\mathbb{Q}$.

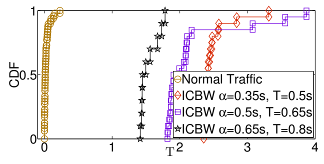


(d) Detect SWIRL through $\mathbb{Q}$.

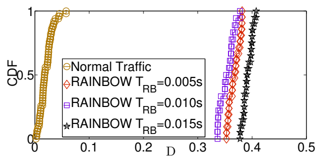
Figure 5: The distributions of metrics $\mathbb{R}$ and $\mathbb{Q}$ in normal HTTP flows and those in HTTP flows watermarked by IBW, ICBW, RAINBOW and SWIRL using different settings.



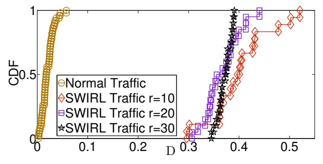
(a) Detect IBW through $\mathbb{T}$.



(b) Detect ICBW through $\mathbb{T}$.



(c) Detect RAINBOW through $\mathbb{D}$.



(d) Detect SWIRL through $\mathbb{D}$.

Figure 6: The distributions of metrics $\mathbb{T}$ and $\mathbb{D}$ in normal SSH flows and those in SSH flows watermarked by IBW, ICBW, RAINBOW and SWIRL using different settings.

ployment of traffic watermarking schemes. RRT samples are obtained by sending requests to the server and then recording the timestamp of a request's last packet (if a request consists of more than one packet) and the timestamp of the response's first packet. The response packet's acknowledgement number should be equal to the summation of the request packet's sequence number and its payload length. We compute IPDs from the response packets triggered by each single request. We calculate burst size in a flow using the method outlined in [19].

Since packet loss may lead to biased RRT, IPD and burst size samples, we exploit TCP's basic mechanism to filter out potentially biased samples that match one of the following criteria:

1. Samples containing out-of-order data packets. Nowadays packet reordering due to network elements is not prevalent, and most reordered packets are caused by packet retransmission [25].
2. Samples including data packets that follow three duplicate ACK packets. These ACK packets suggest that a TCP sender will use the fast retransmit/fast recovery mechanism to retransmit lost packets [26].
3. Samples comprising duplicate packets. This may happen when ACK packets are lost and then the server consequently retransmits the unacknowledged data packets.
4. IPDs that are close to or larger than the server's retransmission timeout (RTO) value and are followed by retransmitted packets. This rule is also applied to RRTs that are close to or larger than the summation of RTO and ξ_{RTT}. Even if there are not retransmitted packets, we can still infer whether a packet is retransmitted through the identification field in the IP header (IPID) or the estimated congestion window (cwnd) [27]. These methods are also employed to determine whether a one-packet burst is due to packet loss. The server's features, such as RTO, IPID, and cwnd can be measured using `TBIT`'s method [28].

5. IMPLEMENTATION

We implement IBW, ICBW, RAINBOW and SWIRL on Ubuntu Linux (kernel 2.6.35-22) using `iptables` (version 1.4.4) and `libnetfilter_queue` library (version 1.0.0). We add rules into `iptables`'s OUTPUT chains to hook outgoing TCP packets from the service under surveillance (e.g., HTTP, SSH). These packets are queued in the kernel, and our program acquires them by invoking `libnetfilter_queue`. By exploiting the head-of-blocking feature of the system's output queue, our program can delay a batch of packets by holding the first packet for a designated time period.

For IBW, we implement the interval selection function that chooses every second and third intervals to embed information [2]. For an ICBW interval, our algorithm randomly determines, with a probability of 0.5, whether packets within that interval will be delayed or not, because ICBW randomly selects intervals and then delays packets in selected intervals [3]. For SWIRL, we generate the permutation randomly, because Houmansadr et al. do not describe how to construct π in [7]. For RAINBOW, we follow [6] to prepare the delay added to each packet in advance and set the upper bound of T_0 to 50ms. Since in practice RAINBOW cannot predict when the next packet will arrive, we add pre-calculated delay to each packet independently. It is worth noting that our implementation does not allow reordered packets (i.e., if the adjusted IPD is less than zero, it will be set to zero), because frequent packet reordering is suspicious. Moreover, reordered packets cannot carry watermarks through stepping stones and anonymity networks, because the TCP/IP stack in a relay node will rearrange reordered packets and send *in-order* packets to the next node.

BACKLIT comprises a set of Python scripts and Matlab scripts that carry out the entire detection procedure. It instructs hosts to visit the target server through HTTP or SSH, extracts metrics from the observed traffic, and performs the detection. The detection module is based on the DD_tools Matlab toolbox 1.7.3 [29] and the PRTools Matlab toolbox 4.1.4 [30]. We employ the OC classifier based on support vector data description (SVDD) and detailed information about OC classifiers can be found in [23].

6. EVALUATION

6.1 Experiment settings

As shown in Table 4, most of the previous research on traffic watermarking used replayed or synthetic traffic for evaluation purposes [2, 6, 7, 12], with the exceptions of ICBW [3] and PNR [11] which employed live traffic. The use of replayed or synthetic traffic is inadequate because they cannot legitimately represent TCP's behaviors, such as being responsive to network conditions and adapting its parameters to the changes. Moreover, replaying packets in either one or two directions of interactive traffic cannot mimic the real environment, because the RTT and network conditions on the path between two hosts in the trace may not be the same as those on the path between the two hosts replaying the traffic.

We used live HTTP and SSH flows to evaluate BACKLIT. The four watermarking schemes were deployed on a host in Hong Kong to embed watermarks into HTTP and SSH connections between the server and 12 PlanetLab nodes around the world, which are listed in Table 5.

For experiments using HTTP traffic, `curl` was run with the option `-max-time` on each PlanetLab node to download a large file from the server for 90 seconds. For experiments using SSH traffic, we implemented an SSH client based on `libssh2` [31] and ran it on the PlanetLab nodes. Besides using the default settings suggested in [2, 3, 6, 7], we also adopted parameters in line with the rules suggested by those traffic watermarks. Table 6 summarizes the parameter settings used in the experiments.

6.2 Evaluating BACKLIT's false positive rate

To evaluate BACKLIT using HTTP traffic, each PlanetLab node first downloaded a large file 60 times. Half of the traces were used to estimate ξ_{RTT} and train the OC classifier. The remaining traces were employed to evaluate BACKLIT's false positive rate. When evaluating BACKLIT using SSH, we executed a sequence of non-existent commands (i.e., input "a" and return). This is a simple way to test the presence of traffic watermarks. We observed that sending "a" and the return character triggers four packets from the server. The first two packets echoed "a" and the return character. The other two packets may contain the error message (e.g., `-bash: a: command not found`). Before executing a new command, BACKLIT slept for 100ms. We executed the commands for 60 times in each PlanetLab node. Half of the traces were used to train the OC classifier and the remaining traces were employed to evaluate BACKLIT's false positive rate.

We first set P_R to 0 and then 0.03. The former indicates that all normal samples are used to build the detection metrics profiles, and the latter means that all except one normal sample are employed, because the training data set consists of 30 samples. Table 7 shows the false positive rates for detection metrics $\mathbb{R}$, $\mathbb{Q}$, $\mathbb{T}$, and $\mathbb{D}$. We did not observe any false positive in $\mathbb{Z}$.

For $P_R = 0$, none of the metrics produces false positives. For $P_R = 0.03$, the metrics may incorrectly identify at most one out of 30 normal samples as a watermarked one. One possible reason for the false positive is that the sample excluded from the training data due to $P_R = 0.03$ is not an outlier and should be kept. Another possible reason is that the boundary determined by the SVDD OC classifier is too conservative. This is supported by the fact that some misclassified samples from the testing data set were close to the boundary, whereas samples from watermarked flows were far away from the boundary. To remedy this problem, we added artificially generated outlier data to the training data set, labeled them as normal data and then re-trained the OC classifier. This approach helped remove the false positives. The DD_tools toolbox provides functions, such as `gendatout`, to generate such outliers based on existing data.

6.3 Evaluating BACKLIT's detection rate

To evaluate BACKLIT's detection rate, we activated each traffic watermarking scheme one by one and then asked each PlanetLab node to download the big file 20 times for acquiring watermarked HTTP traces. To collect watermarked SSH traces, BACKLIT instructed each PlanetLab node to enter the SSH commands 20 times. Setting $P_R = 0.03$ to exclude potential outliers in the training data set, BACKLIT discovered *all* the watermarked HTTP flows and only missed two watermarked SSH flows. One was an SSH flow watermarked by SWIRL using parameter setting 1 on the path from a PlanetLab node in France and the other was an SSH flow watermarked by IBW using parameter setting 1 on the path from a PlanetLab node in Japan (i.e., JP3).

The detection rates for IBW and ICBW are high, because they delay a set of packets for a long period (i.e., several hundred milliseconds) to facilitate the decoding of traffic watermarks [2,3]. By exploiting TCP's basic mechanism, we can filter out long IPDs resulting from packet loss. Such IPDs are similar to the delay introduced by IBW or ICBW and may cause false positives. Although MFA also exploits the long delay caused by IBW and ICBW, it did not mention whether such noises were removed and did not report the false positive rate [12]. We attribute the high detection rates for RAINBOW and SWIRL to the fact that they aggressively affect contiguous packets, although the introduced delay is small in comparison with IBW and ICBW. In particular, RAINBOW affects almost every packet, while SWIRL influences packets in consecutive short intervals.

After investigating the undetected watermarked flows, we found that both SSH connections were broken up after BACKLIT dispatched a few "a" commands. It may be caused by the instability of the PlanetLab nodes or network congestions. In the flow watermarked by IBW only part of the response packets were delayed and the period was short. The reason is that IBW only delays packets in selected intervals and if a packet is close to the end of a selected interval it will be delayed for a short period. Similarly, SWIRL only postponed some response packets and the $|\Delta\text{RRT}|$s affected by SWIRL were not obvious. It may be due to network congestions or cross traffic that could have delayed other response packets. Since existing timing-based traffic watermarking schemes usually require several hundreds or thousands of packets to embed traffic watermarks that can be successfully decoded [2, 3, 6, 7], it is difficult for them to trace short flows. On the other hand, BACKLIT can increase the sending rate of requests (i.e., in the active mode) to gain more prolonged RRT samples.

7. RELATED WORK

7.1 Timing-based traffic watermarks

Tracing network communications is motivated by a well-known security problem, namely detecting stepping-stone attacks. The pioneering work from Staniford-Chen and Heberlein used the thumbprint of a packets' content to correlate flows [32]. With the prevalence of encrypted payloads, content-based approaches have become less effective [32]. Zhang and Paxson exploited *invariant* traffic features, instead of the packet content [18], to detect stepping-stone attacks. Their method belongs to a class of passive approaches that do not perturb the traffic under surveillance. Besides the ON/OFF patterns employed in [18], many other features have been examined in recent papers [33,34], for example, packet count, inter-packet delay and the increasing pattern of TCP sequence number, to name a few. However, these approaches are vulnerable to normal time perturbations and *chaff* packets, and usually require thousands of packets to be effective [6].

Wang et al. proposed an original method that actively embeds watermarks into a flow by adjusting the timing of randomly selected packets [1]. Recently, more advanced flow watermarks have been proposed, which used sophisticated approaches to manipu-

Table 4: Experiment traffic used in this paper, IBW [2], ICBW [3], RAINBOW [6], SWIRL [7], PNR [11], and MFA [12].

Traffic	Synthetic SSH	Replayed SSH	Replayed HTTP	Live SSH	Live HTTP
Watermarking Schemes	IBW, ICBW	RAINBOW, SWIRL			ICBW
Detection Mechanism		MFA	MFA	PNR[†], **BACKLIT**	**BACKLIT**

[†] PNR does not indicate the traffic type, which might be live SSH traffic as the experiment was conducted between stepping stones.

Table 5: The 12 PlanetLab nodes used in the experiments.

Country	IP	Country	IP	Country	IP	Country	IP	Country	IP	Country	IP
FR	132.227.62.25	CH	130.92.70.254	SG 1	137.132.80.106	SG 2	203.30.39.238	FI	193.166.167.5	JP 1	133.68.253.243
JP 2	202.23.159.52	JP 3	150.65.32.68	NZ	132.181.10.57	DE	141.20.103.211	BR	200.17.202.195	US	208.94.63.193

Table 6: Parameters used for each type of traffic watermark.

	IBW	ICBW	SWIRL	RAINBOW
Setting 1	$T_{IBW} = 0.5$s	$\alpha_{ICBW} = 0.35$s, $T_{ICBW} = 0.5$s[†]	$T_{SWIRL} = 2$s, $m = 5, r = 30$	$T_{RB} = 0.005$s
Setting 2	$T_{IBW} = 0.7$s	$\alpha_{ICBW} = 0.5$s, $T_{ICBW} = 0.65$s	$T_{SWIRL} = 2$s, $m = 5, r = 20$	$T_{RB} = 0.01$s
Setting 3	$T_{IBW} = 0.9$s[†]	$\alpha_{ICBW} = 0.65$s, $T_{ICBW} = 0.8$s	$T_{SWIRL} = 2$s, $m = 5, r = 10$	$T_{RB} = 0.015$s

[†] The experiment settings used for evaluating MFA [12].

Table 7: False positive rates obtained from the PlanetLab nodes for HTTP and SSH traffic.

Country	FR		CH		SG 1		SG 2		FI		JP 1		JP 2		JP 3		NZ		DE		BR		US	
P_R	0	0.03	0	0.03	0	0.03	0	0.03	0	0.03	0	0.03	0	0.03	0	0.03	0	0.03	0	0.03	0	0.03	0	0.03
$\mathbb{R}$ (HTTP)	0	0.03	0	0.03	0	0.03	0	0	0	0	0	0.03	0	0.03	0	0	0	0	0	0	0	0	0	0
$\mathbb{Q}$ (HTTP)	0	0.03	0	0	0	0.03	0	0.03	0	0	0	0	0	0.03	0	0	0	0.03	0	0	0	0.03	0	0.03
$\mathbb{T}$ (SSH)	0	0	0	0.03	0	0	0	0.03	0	0	0	0.03	0	0	0	0	0	0	0	0.03	0	0	0	0
$\mathbb{D}$ (SSH)	0	0	0	0	0	0.03	0	0.03	0	0.03	0	0	0	0.03	0	0	0	0.03	0	0.03	0	0	0	0

late packet timing information [2, 3, 6, 7] or the traffic rate [35]. Houmansadr et al. [6] classified watermarking schemes into *blind* and *non-blind*, depending on whether the decoder has *a priori* information about the flow that is being watermarked. Most existing watermarking techniques belong to the blind scheme category [1–3, 5, 35], whereas RAINBOW [6] is a non-blind scheme.

7.2 Countermeasures

Although various traffic watermarks have been designed, only a few approaches have been proposed to detect them. PNR [11], a pioneering work from Peng et al., employs inflated one-way packet delays to detect the traffic watermarks proposed in [1] and estimates its parameters. However, PNR adds a timestamp to each packet and has to handle the synchronization issues. BACKLIT does not insert information into the packets or involve the cooperation of a remote server, although this can help detection. Instead, BACKLIT exploits TCP's timing features for the detection.

Kiyavash et al. proposed an interesting detection scheme, MFA [12], which exploits the observation that when the same watermarks are embedded to multiple flows simultaneously, there will be abnormally long idle periods in the aggregated traffic. However, MFA can be evaded if the encoder randomizes the location of the watermarks or uses different watermarks on different flows, as reported in their follow-on paper [15]. Moreover, MFA requires multiple flows for successful detection and assumes that normal traffic follows the Markov-modulated Poisson process model. BLACKLIT can detect watermarks in a single flow and does not make any assumption about the distribution of normal traffic. Moreover, BLACKLIT is not affected by the random location of watermarks.

Yu et al. devised an invisible throughput-based traffic watermarking scheme that uses the direct-sequence spread spectrum theory to hide the watermarks [35]. We proposed an approach exploiting the features of Pseudo-Noise (PN) codes to detect such watermarks [36].

8. CONCLUSION

Traffic watermarking is important to many network security and privacy applications, because it can correlate network flows observed at different locations quickly and accurately. The state-of-the-art timing-based traffic watermarks can not only survive adversarial network conditions but also evade existing detection methods. However, we show in this paper for the first time that all of them will disturb the intrinsic timing features in TCP flows and propose a novel system, BACKLIT, to detect them. Our extensive empirical evaluation has shown that BACKLIT can detect the state-of-the-art traffic watermarks with high detection rate and low false positive rate. By exploiting these TCP features, BACKLIT can potentially be extended to detect other timing-based traffic watermarks. In future work, we will differentiate these traffic watermarks based on different anomalies and exploit the disturbed features to estimate their parameters.

Acknowledgments

We thank the anonymous reviewers and Dr. John McDermott for their very useful suggestions and Michael Cumming for editing the paper. This work is partially supported by a grant (G-U386) from The Hong Kong Polytechnic University. This material is based upon work supported in part by the National Science Foundation under grant no. 0831300, the Department of Homeland Security under contract no. FA8750-08-2-0141, the Office of Naval Research under grants no. N000140710907 and no. N000140911042. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation, the Department of Homeland Security, or the Office of Naval Research.

9. REFERENCES

[1] X. Wang and D. Reeves, “Robust correlation of encrypted attack traffic through stepping stones by manipulation of interpacket delays,” in *Proc. ACM CCS*, 2003.

[2] Y. Pyun, Y. Park, X. Wang, D. Reeves, and P. Ning, “Tracing traffic through intermediate hosts that repacketize flows,” in *Proc. IEEE INFOCOM*, 2007.

[3] X. Wang, S. Chen, and S. Jajodia, “Network flow watermarking attack on low-latency anonymous communication systems,” in *Proc. IEEE Symp. Security and Privacy*, 2007.

[4] D. Ramsbrock, X. Wang, and X. Jiang, “A first step towards live botmaster traceback,” in *Proc. RAID*, 2008.

[5] X. Wang, S. Chen, and S. Jajodia, “Tracking anonymous peer-to-peer voip calls on the Internet,” in *Proc. ACM CCS*, 2005.

[6] A. Houmansadr, N. Kiyavash, and N. Borisov, “RAINBOW: A robust and invisible non-blind watermark for network flows,” in *Proc. NDSS*, 2009.

[7] A. Houmansadr and N. Borisov, “SWIRL: A scalable watermark to detect correlated network flows,” in *Proc. NDSS*, 2011.

[8] V. Shmatikov and M. Wang, “Timing analysis in low-latency mix networks: attacks and defenses?” in *Proc. ESORICS*, 2006.

[9] H. Daginawala and M. Wright, “Studying timing analysis on the Internet with SubRosa,” in *Proc. PET*, 2008.

[10] W. Wang, M. Motani, and V. Srinivasan, “Dependent link padding algorithms for low latency anonymity systems,” in *Proc. ACM CCS*, 2008.

[11] P. Peng, P. Ning, and D. Reeves, “On the secrecy of timing-based active watermarking trace-back techniques,” in *Proc. IEEE Symp. Security and Privacy*, 2006.

[12] N. Kiyavash, A. Houmansadr, and N. Borisov, “Multi-flow attacks against network flow watermarking schemes,” in *Proc. USENIX Security*, 2008.

[13] A. Hernandez and E. Magana, “One-way delay measurement and characterization,” in *Proc. IEEE ICNS*, 2007.

[14] N. Macfadyen, “Traffic characterisation and modelling,” *BT Technology Journal*, vol. 20, no. 3, 2002.

[15] A. Houmansadr, N. Kiyavash, and N. Borisov., “Multi-flow attack resistant watermarks for network flows,” in *Proc. IEEE ICASSP*, 2009.

[16] W. John and S. Tafvelin, “Analysis of Internet backbone traffic and header anomalies observed,” in *Proc. ACM/USENIX IMC*, 2007.

[17] R. Dingledine, N. Mathewson, and P. Syverson, “Tor: The second-generation onion router,” in *Proc. USENIX SECURITY*, 2004.

[18] Y. Zhang and V. Paxson, “Detecting stepping stones,” in *Proc. USENIX Security*, 2000.

[19] S. Shakkottai, N. Brownlee, and k. claffy, “A study of burstiness in TCP flows,” in *Proc. Passive and Active Measurement Conf.*, 2005.

[20] H. Jiang and C. Dovrolis, “Passive estimation of TCP round-trip times,” *ACM Computer Commun. Review*, 2002.

[21] C. Dovrolis, P. Ramanathan, and D. Moore, “Packet dispersion techniques and a capacity-estimation methodology,” *IEEE/ACM Trans. Networking*, vol. 12, no. 6, 2004.

[22] C. Therrien and M. Tummala, *Probability for Electrical and Computer Engineers*. CRC, 2004.

[23] D. Tax, “One-class classification; concept-learning in the absence of counter-examples,” Ph.D. dissertation, Delft University of Technology, 2001.

[24] R. Perdisci, G. Gu, and W. Lee., “Using an ensemble of one-class SVM classifiers to harden payload-based anomaly detection systems,” in *Proc. IEEE ICDM*, 2006.

[25] L. Gharai, C. Perkins, and T. Lehman, “Packet reordering, high speed networks and transport protocol performance,” in *Proc. IEEE ICCCN*, 2004.

[26] M. Allman, V. Paxson, and W. Stevens, “TCP congestion control,” RFC 2581, April 1999.

[27] S. Jaiswal, G. Iannaccone, C. Diot, J. Kurose, and D. Towsley, “Measurement and classification of out-of-sequence packets in a tier-1 IP backbone,” *IEEE/ACM Trans. Networking*, vol. 15, no. 1, 2007.

[28] A. Medina, M. Allman, and S. Floyd, “Measuring the evolution of transport protocols in the Internet,” *ACM Computer Communication Review*, 2005.

[29] D. Tax, “DDtools, the data description toolbox for Matlab (version 1.5.3),” June 2009.

[30] “PRTools: The Matlab toolbox for pattern recognition,” http://www.prtools.org/, 2008.

[31] “libssh2,” http://www.libssh2.org/, 2011.

[32] L. Heberlein and S. Staniford-Chen, “Holding intruders accountable on the Internet,” in *Proc. IEEE Symp. Security and Privacy*, 1995.

[33] A. Blum, D. Song, and S. Venkataraman, “Detection of interactive stepping stones: Algorithms and confidence bounds,” in *Proc. RAID*, 2004.

[34] B. Coskun and N. Memon, “Efficient detection of delay-constrained relay nodes,” in *Proc. ACSAC*, 2007.

[35] W. Yu, X. Fu, S. Graham, D. Xuan, and W. Zhao, “DSSS-based flow marking technique for invisible traceback,” in *Proc. IEEE Symp. Security and Privacy*, 2007.

[36] X. Luo, J. Zhang, R. Perdisci, and W. Lee, “On the secrecy of spread-spectrum flow watermarks,” in *Proc. ESORICS*, 2010.

APPENDIX

LEMMA 1. *Given a sequence of L observations of IPDs, where L_r IPDs are equal to ξ_{RTT} and the remaining IPDs are equal to ξ_{CAP}, the probability that there will be no consecutive $IPDs = \xi_{RTT}$s is:* $P_{\mathfrak{w}} = \frac{\binom{L-L_r+1}{L_r}}{\binom{L}{L_r}}$.

PROOF. We define a run of $IPDs = \xi_{RTT}$ (or $IPDs = \xi_{CAP}$) as a sequence of length $l \geq 1$ of consecutive IPDs that have a value equal to ξ_{RTT} (or ξ_{CAP}). If there are no consecutive $IPDs = \xi_{RTT}$, then there are L_r runs of $IPDs = \xi_{RTT}$ having length $l = 1$, and the number of runs of $IPDs = \xi_{CAP}$ is equal to $L_r - 1$, $L_r + 1$, or L_r, because the runs of $IPDs = \xi_{RTT}$ and the runs of $IPDs = \xi_{CAP}$ must alternate.

If the value is equal to $L_r - 1$, then the first observation is $IPD = \xi_{RTT}$. If it is equal to $L_r + 1$, then the first observation is $IPD = \xi_{CAP}$. If the value is equal to L_r, either $IPD = \xi_{RTT}$ or $IPD = \xi_{CAP}$ could be the first observation. Since the number of combinations of putting $(L - L_r)$ $IPDs = \xi_{CAP}$ into L_r runs is $\binom{L-L_r-1}{L_r-1}$, the probability that there are L_r runs of $IPDs = \xi_{RTT}$ is equal to $\frac{\binom{L-L_r-1}{L_r}+\binom{L-L_r-1}{L_r-2}+2*\binom{L-L_r-1}{L_r-1}}{\binom{L}{L_r}} = \frac{\binom{L-L_r+1}{L_r}}{\binom{L}{L_r}}$. □

Exploring the Potential Benefits of Expanded Rate Limiting in Tor: Slow and Steady Wins the Race With Tortoise

W. Brad Moore
Georgetown University
Washington, D.C. 20057
wbm@cs.georgetown.edu

Chris Wacek
Georgetown University
Washington, D.C. 20057
cwacek@cs.georgetown.edu

Micah Sherr
Georgetown University
Washington, D.C. 20057
msherr@cs.georgetown.edu

ABSTRACT

Tor is a volunteer-operated network of application-layer relays that enables users to communicate privately and anonymously. Unfortunately, Tor often exhibits poor performance due to congestion caused by the unbalanced ratio of clients to available relays, as well as a disproportionately high consumption of network capacity by a small fraction of filesharing users.

This paper argues the very counterintuitive notion that *slowing down traffic on Tor will increase the bandwidth capacity of the network* and consequently improve the experience of interactive web users. We introduce Tortoise, a system for rate limiting Tor at its ingress points. We demonstrate that Tortoise incurs little penalty for interactive web users, while significantly decreasing the throughput for filesharers. Our techniques provide incentives to filesharers to configure their Tor clients to also relay traffic, which in turn improves the network's overall performance. We present large-scale emulation results that indicate that interactive users will achieve a significant speedup if even a small fraction of clients opt to run relays.

Categories and Subject Descriptors

C.2.4 [**Computer-Communication Networks**]: Distributed Systems; C.2.0 [**Computer-Communication Networks**]: General—*Security and Protection*; C.2.1 [**Network Architecture and Design**]: [Distributed Networks]

General Terms

Performance, Anonymity, Security

Keywords

Anonymity, Tor, Performance

1. INTRODUCTION

Anonymity networks such as Tor [9] allow their users to privately communicate without revealing their identities. These systems are regularly used to enable private browsing, circumvent censorship firewalls, and provide unfettered access to information. In particular, Tor enables any application that communicates using TCP to tunnel its connections through the anonymity network.[1] This flexibility permits a variety of applications – web browsers, instant messaging clients, file sharing applications, and more – to achieve some degree of anonymity. Tor does not discriminate against any particular application, and moreover, its anonymity features aggravate efforts to distinguish between applications.

A consequence of Tor's versatility is that the anonymity network's capacity is disproportionately consumed by a small subset of users who run high-bandwidth applications. By analyzing the traffic that exited their exit relay in 2008, McCoy *et al.* found that while nearly 97% of observed connections could be classified as *interactive* (e.g., web browsing), approximately 40% of anonymous traffic belonged to *non-interactive* applications [17] such as BitTorrent. By itself, this disproportionate bandwidth utilization is not problematic: Tor is a general-purpose anonymity network, and file sharing has many legitimate uses.

Unfortunately, relative to unprotected communication, Tor suffers from high-latency and low-bandwidth. The network's poor performance not only negatively impacts the applications that it services, it also likely discourages the network's use as many would-be users may be unwilling to sacrifice so much performance for increased privacy. In their performance analysis of Tor, Dingledine and Murdoch identify BitTorrent as a major cause of Tor's slowness [10].

A simplistic approach to improving Tor's performance is to disallow BitTorrent on Tor. However, such a policy arguably runs counter to Tor's philosophy and mission as an anti-censorship technology. Additionally, it is unclear how client applications can be reliably differentiated (though notably, previous work has shown that applications can be probabilistically identified by examining their traffic patterns [13]).

An alternative approach to improving Tor's performance is to increase the number of relays that forward anonymous traffic [10, 14, 22]. A recent study of the Tor network estimates that the number of clients outnumbers the number of available relays by a factor of nearly 67 [15]. Increasing the number of relays diminishes this imbalance and consequently decreases congestion in the network.

This paper adopts this latter technique and proposes an incentive scheme to increase the number of relays on the Tor network and improve the network's overall performance and capacity. Our solution, which we call Tortoise, takes the counterintuitive and seem-

[1]Applying Tor without carefully considering the application's protocol and communication characteristics risks exposing the sender's identity [4, 7]. For example, certain BitTorrent clients annotate requests with their senders' network addresses [4]; similarly, improperly configured end-hosts may reveal receivers' identities by failing to anonymize DNS resolution requests.

ingly contradictory approach that *slowing down Tor will help achieve speedup*. In particular, Tortoise imposes strict rate limits on individual client connections at the network's ingress points but does not limit connections originating from Tor relays. By carefully tuning these limits, interactive clients such as web browsers see little effect, while bandwidth-intensive users (for example, filesharers) experience a significant decrease in throughput. We argue that this slowdown provides incentives for filesharers to operate their own relays through which they can bypass the network's rate limiting. We posit that some fraction of filesharers will become sufficiently frustrated to operate their own relays, which will in turn serve additional traffic and reduce congestion. Moreover, the savings in bandwidth from rate limiting filesharers and other high-throughput users who do not run relays can provide additional network capacity.

Unlike recently proposed incentive and e-cash approaches that require centralized *mints* [14] or *banks* [2, 5], our solution is fully decentralized, is backwards-compatible with the existing Tor network, and may be deployed incrementally. We demonstrate the feasibility of Tortoise and investigate its ability to improve the performance of interactive web clients through emulation using ExperimenTor [3], a large-scale network emulator that executes actual Tor binaries on a virtual network. We show that the performance of interactive web browsers will significantly increase if just a small number of clients opt to run their own relays.

Threat Model and Limitations. Importantly, Tortoise is not robust against determined adversaries who wish to circumvent its rate limiting through Sybil-style attacks [11]. For example, an attacker can achieve high throughput by connecting to many (rate limited) relays and aggregating their bandwidths. Although existing Sybil defenses may offer some protection, we note that Tortoise does not worsen the performance of the network (relative to unmodified Tor) in the presence of such determined attackers. Rather, Tortoise *is designed to provide incentives for ordinary users* – some of whom desire high throughputs – to also operate as relays. If a small fraction of such honest users opt to relay Tor traffic, then the network will improve. More sophisticated high-bandwidth users may evade the rate limits, but as we discuss below, doing so may be more costly than behaving correctly and operating a relay.

Additionally, as with other solutions that motivate users to run relays, Tortoise inherently trades off performance for anonymity: operating a relay increases performance, but decreases sender anonymity since the initiator of high-throughput traffic is likely the operator of a Tor relay. We discuss the security implications of using Tortoise in more detail below.

We begin by reviewing the Tor network and describing its current rate limiting features.

2. BACKGROUND

Tor is a volunteer-operated network of approximately 2,500 application-layer routers (also called *relays* or ORs). The network provides anonymity by forwarding traffic from clients (also called *proxies* or OPs) along a bidirectional *anonymous circuit* consisting of Tor routers. To conceal the identities of the communicants, Tor encrypts messages such that each relay can discern only the identities of the previous and next hops along the anonymous circuit. By default, Tor uses three-relay hops, consisting of a guard relay, a middle relay, and an exit relay.

Increasing the number of relays provides greater anonymity and performance. The number and configuration of Tor relays determine the network's performance and anonymity. A large number of (honest) relays provides strong anonymity since traffic has a lower likelihood of traversing only malicious relays. (If the guard and exit relays are malicious and colluding, then the adversary can identify the sender and receiver of intercepted communication [19, 32].) In addition, an increase in the number of Tor relays improves the performance of the network by providing additional capacity, which in turn decreases congestion.

Ideally, each person who uses Tor would also run a Tor relay, contributing some bandwidth to the network's overall capacity. Unfortunately, the current ratio of end-clients to relays is estimated to be 67:1 [15], leading to significant congestion and poor performance. This imbalance can be partially explained by the multiple costs of running a Tor relay: Operating a relay taxes both the hosting computer as well as its network connection. In order to provide a benefit to the network, a router must be continuously online for weeks before Tor clients will begin to use it. Additionally, when Tor is configured to operate as an exit router, the operator's computer may appear to law enforcement officials to be accessing illegal content. Unlike the two other relays that comprise a Tor circuit, the exit relay directly accesses the server requested by the sender; if this service serves illegal content, it will appear to the outside world that the request originated at the exit relay, putting the relay's operator at substantial risk. There are currently few incentives to operating a relay, and it is reasonable to assume that most current Tor relay operators volunteer their computer and network resources for altruistic reasons.

Rate limiting in Tor. Tor includes rate limiting features that allow relay operators to configure how much collective bandwidth they wish to delegate for serving Tor traffic. The existing functionality does not currently support per-connection rate limits, although such features are present in alpha releases. As described below, Tortoise extends Tor's rate limiting by throttling clients' inbound traffic.

3. DISMISSED: FILESHARER IDENTIFICATION AND FILTERING

A seemingly plausible and straightforward method of reducing the strain on the Tor network is to filter filesharing traffic at exit relays using standard port blocking. In fact, the Tor Project recommends that users running exit routers block BitTorrent's default ports [23]. However, such filtering does little to deter determined filesharers since users of these services can trivially switch to non-standard ports.

Alternatively, exit relays could apply more advanced techniques and perform deep packet inspection (DPI) and/or traffic fingerprinting [13] on the traffic that they forward. Recall that once a user's traffic has reached the exit relay, it is no longer protected by any layers of encryption that were applied by Tor (since the exit relay must interface with the destination server as if it were the original client). Hence, exit relays could examine outgoing traffic and discard any detected BitTorrent packets.

However, applying DPI and traffic fingerprinting at exit relays suffers from several shortcomings. First, and perhaps most importantly, such strategies are antithetical to the goals of the Tor project. Tor is an anonymity network whose principal purpose is to provide its users with unfettered Internet access without the fear that their traffic is being monitored. In order to be effective, the traffic blocking schemes described above would necessarily have to violate Tor's underlying philosophy by engineering eavesdropping into the system's design. Relatedly, another of Tor's goals is to allow its users to access content that would otherwise be unavailable to them; actively blocking content is incompatible with this goal. Additionally, if Tor were to attempt to identify and limit cer-

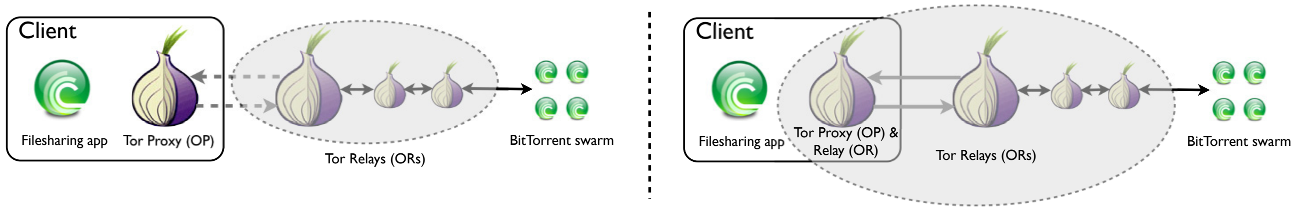


Figure 1: Tortoise's universal rate limiting. Dashed lines indicate connections that are subject to Tortoise's universal rate limit. The shaded circle encompasses relays that comprise the Tor network. *Left:* Client operates a OP and is subject to the universal rate limit. *Right:* Client additionally operates an OR, increasing the size of the Tor network and becoming exempt from the universal rate limit.

tain types of traffic, users generating that traffic could always shape their communication to resemble a type of traffic not easily discriminated against by Tor (for example, communication that is shaped like encrypted web traffic). Finally, performing either DPI or fingerprinting techniques imposes added complexity and increases the relays' computational costs.

At best, identifying filesharers is an arms race: detection approaches will likely be followed and countered by obfuscation techniques, *ad nauseam*. In the next section, we present Tortoise, a universal rate limiting approach that is applied to *all* communication, thereby evading this adversarial arms race.

4. TORTOISE

Tortoise modifies Tor's already-implemented (but not very utilized) token-bucket system to limit users' bandwidths at the network's ingress points. Our goal is to establish a *universal rate limit* that imposes a heavy throughput penalty for users who use the network for bulk transfers while not significantly degrading the experience of users who use the anonymity service for interactive web browsing.

By itself, a universal rate limit will do little to improve Tor's performance. Imposing bandwidth limits on bulk transfer users is unlikely to reduce their overall effect on the network, since their use of the Tor network already indicates their willingness to tolerate slow speeds. For instance, halving their speeds with a universal rate limit is likely only to double the time the bulk users spend on the network. In general, merely penalizing bulk transfer users is a zero-sum game.

An intuitive strategy is to apply a low universal rate limit to provide incentives to all users (both low- and high-bandwidth clients) to operate Tor relays. However, many users of the Tor network connect from totalitarian regimes where Internet access is severely limited and subject to strict censorship. Requiring these users to operate relays not only does not add significant capacity to the Tor network, such a policy may also physically endanger the operators. Instead, Tortoise is designed to place the burden on users who require large bandwidths. That is, anyone can access Tor and achieve bandwidth that is suitable for web browsing. Users who require greater bandwidths are incentivized to also offer their services as Tor relays.

Tortoise aims to improve the overall performance of the Tor network not by traffic shaping, but rather by increasing the capacity of the network by encouraging users to run routers. Tortoise achieves this goal by enforcing the universal rate limit only on Tor OPs (clients); the connections between Tor ORs (relays) are not impacted by Tortoise and are subject only to relay-specific bandwidth limits. (As described below, Tortoise requires ORs to meet certain conditions in order to be exempted from the universal rate limit.) Hence, clients who also run routers can use their routers as bridges to the Tor network, bypassing Tortoise's universal rate limit. We posit that some bulk transfer users who find their bandwidth on the Tor network severely limited will be motivated to bypass the bandwidth limits by running their own OR.

An illustration of Tortoise's universal rate limiting is presented in Figure 1. Initially (Figure 1, *Left*), a filesharing client tunnels his traffic through Tor and is subject to Tortoise's universal rate limit (indicated in the Figure with dashed lines). To achieve better performance, the client then opts to also run a Tor relay (Figure 1, *Right*). The new relay increases the size of the Tor network, which consequently decreases congestion and improves the network's overall performance.

4.1 Preventing Cheating

High-bandwidth users who wish to evade the universal rate limit may do so by operating their own relay. However, Tortoise should ensure that those relays are actually contributing to the performance of the Tor network *in toto*. For example, a user could attempt to game the system by running a relay only when it wants to download content at high speed, or it may operate a very low-bandwidth relay that has little effect on the network's overall capacity.

Tortoise mitigates these "cheats" by relying on status flags maintained by the Tor directory servers. To prevent a user from taking advantage of Tortoise by running a router only at times when they want increased performance, Tortoise requires that a router be listed as STABLE in directory servers; connections from all other routers are subject to the universal rate limit. We note that applying rate limits to non-STABLE routers will not significantly impact the performance of the network, since Tor's default relay selection strategy biases selection in favor of STABLE relays. In order to appear as STABLE, a router must have a mean-time-between-failures greater than that of the median of all other routers [30]. At the time of this writing, the 50th percentile of Tor routers had an uptime of approximately four days.

Additionally, to prevent rewarding a user who operates a STABLE relay that offers very little bandwidth to the Tor network, only relays that are marked as FAST in the directory servers are excluded from the universal rate limit. FAST routers are defined as those that offer at least 20KBps or have bandwidths that are in the top 87.5% of known relays [30]. Tor's default relay selection strategy also heavily biases selection towards FAST relays, and hence applying the universal rate limit to non-FAST relays will not significantly degrade the performance of the network.

In summary, relays that are marked as STABLE and FAST are exempt from the universal rate limit. Currently, these are exactly the relays that are selected by Tor's relay selection algorithms, and consequently, are the relays that forward Tor's traffic.

4.2 Anonymity Considerations

At first blush, it may appear that Tortoise allows eavesdroppers to distinguish between encrypted traffic that belongs to a filesharer and that which belongs to a web client by measuring the monitored connection's throughput. However, upon inspection, this tactic will be less effective than anticipated. While filesharers have the most to gain by running a router under Tortoise, the benefits are not confined to filesharers alone. The incentive to run a Tor relay applies to all users who desire faster speeds through Tor, and hence high-bandwidth traffic may belong to any user who runs a relay.

Admittedly, since the goal in rate limit selection is to avoid adversely affecting web browsing clients (see Section 5.1), it is likely that Tortoise will more adversely affect filesharers, placing greater incentives on that population. However, because there are an order of magnitude more web users than file sharers using Tor [17], distinguishing between web and filesharer traffic remains difficult. For example, even if the participation rate (i.e., the rate of users who decide to run routers as a result of Tortoise) is ten times higher for filesharers than it is for web browsing clients, the number of web browsing clients that decide to participate will still be three times that of filesharers.

As with other incentive schemes that reward relay operators with additional bandwidth [2, 5, 14], Tortoise reduces anonymity by forcing a smaller *sender anonymity set* – the set of potential senders for a given anonymous connection. With both standard Tor and Tortoise, any Internet-connected device can use the anonymity system, and hence the sender anonymity set is quite large. However, Tortoise's "differentiated services" (that is, the use of rate limited as well as non-rate limited traffic classes) permit the attacker to reason that intercepted high-bandwidth data originates from a node that is also a router. As more users run relays, the size of the anonymity set will similarly increase, as will the anonymity offered by Tortoise. Of course, any reduction in anonymity can be entirely avoided since a relay operator may always choose to not take advantage of his increased bandwidth.

5. EVALUATION

We evaluate Tortoise using *ExperimenTor* [3], a large-scale network emulator that uses ModelNet [31] to model a network topology. Our ExperimenTor deployment consists of two machines: a *edge node* that runs all Tor relays, directory servers, clients, and web servers, and a *core* that emulates the actual network. The edge node has a 12-core 2.8GHz Xeon processor and runs Linux 2.6.35 and Tor version 0.2.1.28. Our default configuration consists of five authoritative directory servers, 15 relays, 900 clients, and 40 web/file servers. Our setup does not utilize guard relays, all relays are potential exit relays, and directory servers do not relay traffic. Tor was extended to include Tortoise's universal rate limiting extensions.

The core machine has a 2.8GHz Pentium D processor and runs FreeBSD 6.3. We assign bandwidth capacities to the Tor relays by randomly sampling bandwidths that were advertised in the live Tor network's directory servers in May 2011. Since Tor's default relay selection strategy is biased in favor of relays that offer the most bandwidth [9], we select from only the highest 300 bandwidths listed by the Tor directory server in order to achieve speeds that closely resemble those of the real Tor network. (We note that in the live Tor network, the first 300 relays advertise 86.5% of the network's total capacity.) The core node simulates a topology in which latencies between all nodes are less than 10ms.

We assign client bandwidths using two classes of client connections: *residential* and *institutional*. Residential connections are asymmetric – download bandwidth is greater than upload bandwidth (as is the case, for example, with most broadband services). Institutional connections are those where bandwidth is symmetric. We assign residential connections to 90% of the clients. The capacity distribution for client connections was selected by cross referencing 2011 country of origin data for Tor clients (gathered by the Tor Project) with average upload and download speeds for those countries as measured by `NetIndex` [21]. For example, since 20% of Tor client connections originate in the United States, we assign 20% of links the mean U.S. upload and download speeds for 2011. Both residential and institutional clients receive capacities from this distribution, but clients designated as institutional are assigned symmetric connections using the download speed for both upstream and downstream traffic.

We instantiate two types of Tor users: **web clients** periodically request web pages. To model a realistic workload, we determined the sizes of the frontpages of the top 60 websites as reported by Alexa [1] (see Figure 2). Sampling from this distribution, web clients request pages of sizes 106KB, 150KB, 238KB, 496KB, and 992KB, each with equal probability. These values represent the 10th, 30th, 50th, 70th, and 90th percentile of web page sizes, respectively. Additionally, web clients are configured to pause an average of 11 seconds, the median "think time" between requests as measured in a study of web browsing behavior [12], and will take between 9 and 11 minute breaks after 15 minute browsing sessions. In contrast, **bulk clients** model high-bandwidth users and continually download files whose sizes are chosen uniformly at random from 1MB, 2MB, 3MB, 4MB, and 5MB. Keeping with the percentages observed by McCoy *et al.* [17], we select 3% (30) of our clients to be bulk; the remaining 870 are configured to be web clients.

We took precautions to ensure that our physical emulation setup did not introduce any processing or network effects. The edge and the core machines are connected via a dedicated 1Gbps link; the total aggregate network throughput did not exceed 1Gbps, and our experiments are not bound by our ExperimenTor configuration's bandwidth or CPU resources.

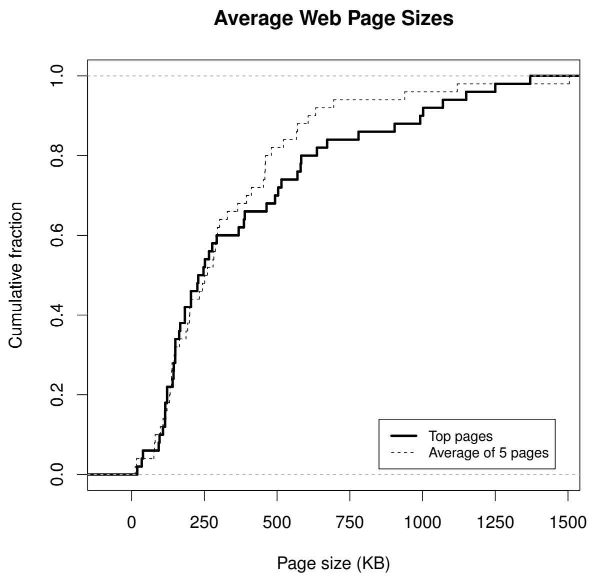


Figure 2: Average page sizes for the top 60 websites as reported by Alexa.

5.1 Rate Limit Selection

The universal rate limit should be sufficiently large to not significantly impact the experience of web users. We can compute a reasonable minimum rate limit by considering both the amount of time that users are willing to wait for a web page to be retrieved, as well as the distribution of web page sizes.

5.2 Effects of Rate Limiting

At the other extreme, the rate limit should not be so high as to allow high-bandwidth users to consume an unfair share of the network's resources. That is, the universal rate limit should be chosen to degrade the performance of high bandwidth users, and consequently provide incentives for them to operate their own relays.

Figure 2 shows the cumulative distribution of the sizes of web pages from the top 60 Alexa [1] web sites. Reported page sizes include embedded images and Javascript but exclude Flash and other content that would not be included in a web page loaded through Tor. The Figure plots both the front page web sizes as well as the average of four additional randomly selected pages on each site.

We select our rate limits based on the expected load time of current web pages. Using the (somewhat dated) heuristic that users tolerate web page load times of eight seconds or less [33], we select a 200KBps limit, which covers all pages from the Alexa dataset. Additionally, we also evaluate Tortoise when using a more stringent 100KBps limit, which covers approximately 80% of the top 60 Alexa websites.

Figure 3 shows the effects of rate limiting at 100KBps (left) and 200KBps (right) on web and bulk clients, when no clients opt to run relays. Web clients are largely unaffected by the rate limit: even with the more severe 100KBps limit, the mean transfer speed drops only 15%, from 40KBps to 34KBps. Bulk transfer clients, on the other hand, are severely affected: even with the less severe 200KBps limit, the mean transfer speed seen by these clients drops 31%, from 70KBps to 49KBps. As we show below, if even a small percentage of the effected clients is sufficiently motivated to operate relays, the overall performance and capacity of the network will improve.

Computational overhead. Tortoise requires that each relay maintain additional token buckets to perform the added rate limiting. Consequently, Tortoise incurs a computational cost relative to unmodified Tor. However, as shown in Figure 4, Tortoise's computational overhead is fairly modest. The Figure plots the CPU utilization of the edge node that executes all 960 Tor instances. Although the edge node's CPU usage is a coarse-grained measure, comparing the processing cost when using unmodified Tor and Tortoise provides a useful estimate of the latter's overhead. With unmodified Tor, the median CPU utilization on ExperimenTor is 24.7%; with Tortoise, the usage increases slightly to 26.4%.

5.3 Performance Improvements

Benefits of adding more relays. We begin by examining the effects of adding more relays to a Tor network. Figure 6 plots the cumulative distribution of client bandwidths when no rate limiting is applied and relays are added to the network. Here, we model an idealized setting in which clients altruistically decide to become relays. It is important to note that such clients generally contribute *less* to the network than the original 15 ORs since (i) the former are generally on slower (e.g., consumer broadband) connections than the dedicated ORs and (ii) as clients, they also use their capacities for their own purposes.

As expected, the more relays that are added, the greater the bandwith available to nodes on the network. In the base case with 15 ORs (1.7% of all nodes), the mean client bandwidth is 41KBps. To determine the effect of clients who also opt to become ORs, we examine two scenarios: in the first case, 10% (2) of bulk transfer clients and 2% (18) of web browsing clients opt to run routers, adding an additional 20 routers to the network. In the second case, these percentages double for an additional 40 routers on the network. In the first configuration, the mean bandwidth experienced by clients increases by 27% to 52KBps. With 40 additional relays, the mean bandwidth grows by 66% to 68KBps.

Though the results of having client Tor instances also run as relays are (unsurprisingly) positive, it is unlikely that, in practice, so many clients will suddenly and unselfishly choose to become routers. *The challenge is to motivate clients to also act as relays*, despite the costs involved. Applying the universal rate limit supplies such motivation, as bulk clients witness their average bandwidths decrease by 31% and 30% with respective limits of 200KBps and 100KBps. (In contrast, web clients incur only 15% and 17% decreases with the two rate limits.) Given the option of achieving greater speeds by running relays, we anticipate that at least a small fraction of bulk (and potentially web) clients will also run ORs. We investigate the advantages of behaving as both an OP and OR below.

Benefits to clients that become relays. Operating an OR exempts a client from Tortoise's universal rate limit. As illustrated in Figure 5, clients that choose to additionally run as an OR will experience better bandwidth than those in the same network that do not. Here, Tortoise utilizes a 100KBps universal rate limit; the low and high adoption rates correspond to the scenarios described above in which 10% (2%) and 20% (4%) of bulk (web) clients opt to run ORs. Although all clients who run ORs experience increased bandwidth, bulk relays gain the greatest benefit from choosing to run a relay since they are most affected by the universal rate limit. Under a 100KBps limit, the mean bandwidth experienced by all Tor clients who chose to run routers increased by 38% and 78% when 10% and 20% of bulk clients became routers, respectively. Most of this improvement was realized by the bulk transfer clients.

Benefits of adding more relays with rate limiting. Figure 7 examines the impact of running Tortoise on network performance. The grey line denotes the average bandwidth seen on a network running unmodified instances of Tor. The solid black line shows the bandwidth of a network using Tortoise. If no clients opt to run as a relay, adopting a 200KBps or 100KBps rate limit will lead to decreased capacity seen by the network. However, given the motivation to run as a relay (see Figure 5), we argue that at least some small fraction of clients will decide to additionally operate as ORs. If only 10% of bulk and 2% of web clients (2.22% of all clients) elect to run relays, the mean bandwidth increases slightly by 3.1% and 0.7% with 100KBps and 200KBps limits, respectively. The addition of another 20 client relays (4.44% of all clients) produces more significant gains, providing 36% and 31% greater mean bandwidths under the 100KBps and 200KBps rate limits respectively, than the unmodified network. As more clients choose to become relays (and hence gain better performance themselves), the network achieves greater speedups.

Handling increasing capacity. Since the "freed" capacity that is not being used due to rate limiting may be applied to serve other clients, a network running Tortoise will be better able to handle additional clients than a network which uses unmodified Tor. Figure 8 depicts the change in network capacity when the sizes of both a Tortoise network and an unmodified Tor network are increased by 20%. (Here, we conservatively assume that no additional nodes run ORs.) The average bandwidth across the network decreases from 45KBps to 30KBps (35%) for the unmodified network, and from 50KBps to 36KBps (28%) on a network running Tortoise with a

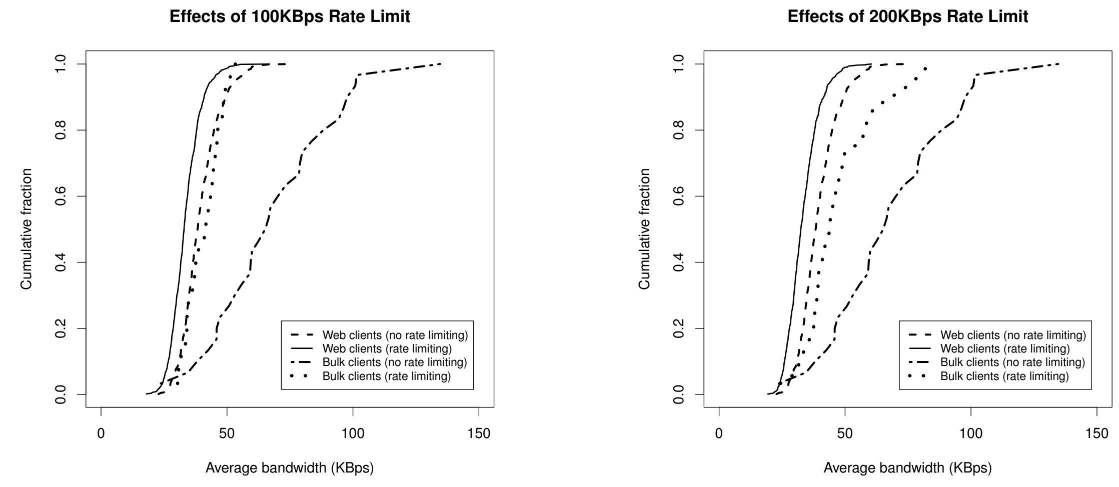


Figure 3: The effects of 100KBPs (left) and 200KBps (right) universal rate limits on web and bulk clients.

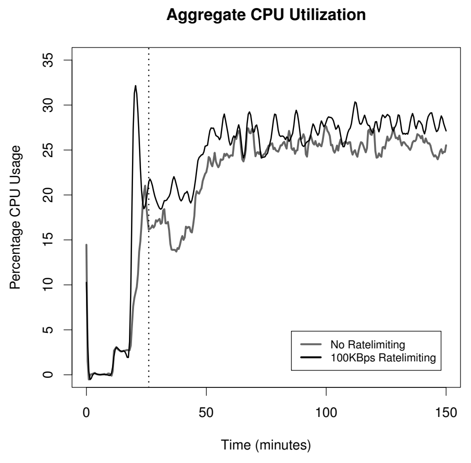


Figure 4: Aggregate CPU usage of the entire network. The vertical dotted line represents the approximate time at which all nodes have joined the network.

100KBps limit. Similarly, with a 200KBps limit, Tortoise's average bandwidth is reduced from 57KBps to 40KBps (30%) when extra clients are added. In other words, the Tortoise-based network is better able to tolerate a sudden and large increase in (non-relay) clients; when the number of clients increases by 20%, the percentage decreases in performance for Tortoise are 20% and 14% less than that of regular Tor when 100KBps and 200KBps rate limits are respectively applied.

Effect of partial participation. To evaluate Tortoise's efficacy when not all relays apply the universal rate limit, we simulated a Tor network in which only 50% of the relays rate limit the clients. Figure 10 shows the cumulative distribution of bandwidths when 0, 20 ("low client adoption"), and 40 ("high client adoption") clients opt to also run relays.

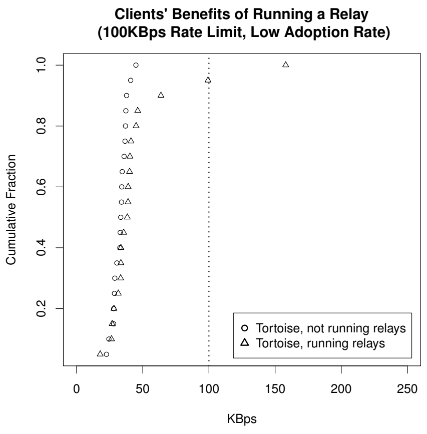
This network's performance was similar to a network with 100% Tortoise adoption in cases where significant numbers of clients chose to run routers: with 20 client routers, this network had a mean bandwidth of 44KBps, vs. 43KBps for the 100% Tortoise network; with 40 routers, these numbers were 54KBps and 56KBps, respectively. The only case where a network with 50% Tortoise adoption showed any significant difference from a 100% Tortoise network was when no clients chose to run routers, in which case the network with 50% adoption exhibited a mean bandwidth of 42KBps, while the 100% Tortoise network averaged 35KBps. While the above data might seem to imply that a network with less than 100% adoption of Tortoise performs better than one with full adoption, it is important to note that a network with less than 100% adoption is less likely to lead to clients choosing to run routers. Figure 9 illustrates the fact that in a network with 50% adoption of Tortoise, there is less motivation to become a router simply because there is less difference between speeds achieved by clients running relays and those not running relays.

5.4 Summary

The above emulation results indicate that Tortoise has the potential to significantly increase Tor's performance. As highlighted above, the challenge of our technique is selecting a universal rate limit that properly motivates clients to operate as relays. If no additional clients serve as relays, then applying a rate limiting trivially slows down the network. However, our emulation results suggest that the addition of even a few relays improves the network's overall performance and capacity, even if the vast majority of clients are subjected to rate limits. Assuming that at least a small number of clients are sufficiently motivated to operate as relays, Tortoise's performance gains can be felt even with partial use of Tortoise rate limits. For instance, the mean client bandwidth increased by 6.5% over standard Tor even in the conservative case when the universal rate limit was 100KBps, only 50% of the relays applied the rate limit, and just 2.2% of clients opted to run relays.

6. DISCUSSION AND LIMITATIONS

Tortoise's relay exemption policy is currently incompatible with bridges. Tortoise may cause problems for *bridges* – unlisted Tor relays that allow users to connect to the Tor network in locations where the public relays are inaccessible. Although bridges may forward traffic from multiple clients, Tortoise will subject them



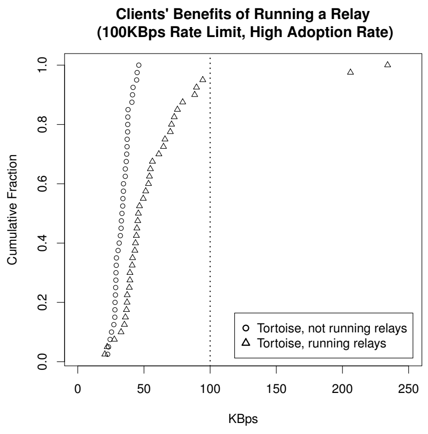


Figure 5: Bandwidth improvements seen by clients that choose to run routers, in a network running Tortoise. The experiments represented in the two graphs differ only in the number of clients who choose to run routers.

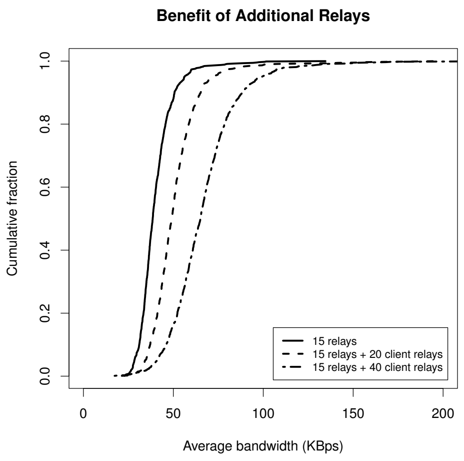


Figure 6: Average bandwidth seen by all clients in an unmodified network, and unmodified networks with additional client relays.

to the universal rate limit since the bridges are not listed in Tor directories.

We propose two methods for adapting Tortoise to better support bridges. For both solutions, we consider a relay that wishes to determine whether an upstream connection is from a client or a bridge: clients should be subject to rate limits, while bridges should be exempt, but only if they are actually forwarding traffic.

One potential approach is for the relay to use a separate bridge to attempt to create a Tor circuit through the node in question. If the circuit is successfully extended, then the node must be a bridge. If the circuit cannot be extended, the node can safely be assumed to be a client. Importantly, since a bridge (known only to the relay) is used to extend the circuit, the node in question cannot behave as a bridge only to the relay (i.e., to be falsely detected as a bridge in order to evade the rate limit). The difficulty with this approach is that if the node is a bridge, its bridge listening port will not be immediately known to the relay. However, the relay can attempt to extend a circuit (via the bridge) using commonly chosen ports.[2] Additionally, Tor would have to be slightly modified to allow bridges to extend circuits to other bridges.

Alternatively, bridges could reveal their identities to independently-chosen trusted relays. These relays would be aware of the bridges' status and will not subject them to the rate limit. The challenge with this approach is that since clients select the anonymous path, bridges would additionally have to recommend a second hop (the trusted relay) that is not subject to rate limiting. We leave the study of these and other potential strategies for supporting Tor bridges as a future research direction.

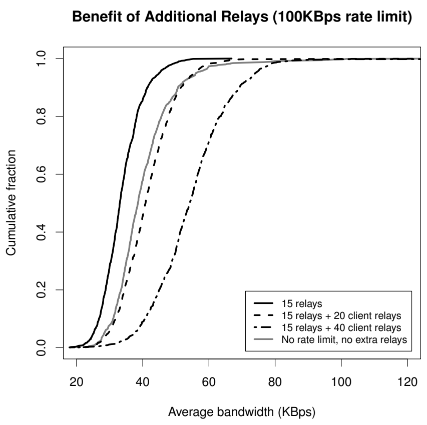
Tortoise relies on accurate directories. Only the relays that are marked STABLE and FAST in the Tor directory are exempt from Tortoise's universal rate limit. A client may attempt to cheat the system by advertising a relay that is neither FAST nor STABLE, but is marked as such by the directory. Since the directories periodically poll the relays to measure their failure rates, a dishonest client cannot easily fake a STABLE rating. It can, however, report a false (high) bandwidth to cause a directory to rate it as FAST.

Several techniques have been recently proposed to avoid the reliance on relays' self-reported capacities. For example, Snader and Borisov introduce an *opportunistic measurement* system in which relays report the observed bandwidth of their peers. Directory servers then advertise the median of these measurements [28]. Perry has suggested an alternative technique in which *measurement authorities* perform empirical measurements of relays' bandwidths [24].

Additionally, a node that wishes to gain exemption from the universal rate limit may contribute only the minimum amount of bandwidth such that they receive the FAST flag. The FAST tag is applied to the fastest 87.5% of routers (as of this writing, this requires a bandwidth of only 15KBps). To better ensure that clients who also run relays are meaningfully contributing to the network, a potential refinement to Tortoise's approach is to additionally apply a rate limit *on relays*. Here, relays' bandwidths would be capped based on the amount of bandwidth that they provide to the network. Hence, the improvement in client bandwidth will be proportional to the amount of bandwidth that the client's OR serves the network.

Tortoise is susceptible to Sybil-style attacks. Tortoise is vul-

[2]Many bridges choose to run on port 443, since HTTPS traffic is often allowed through firewalls, and like HTTPS, Tor uses SSL/TLS to provide confidentiality.



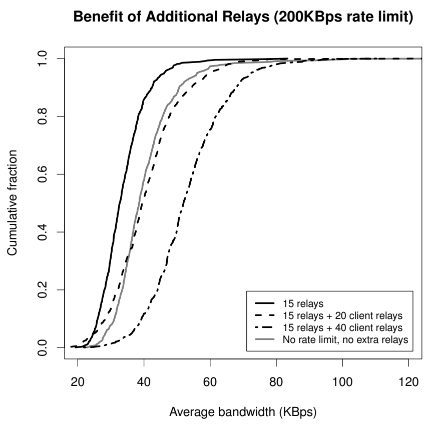
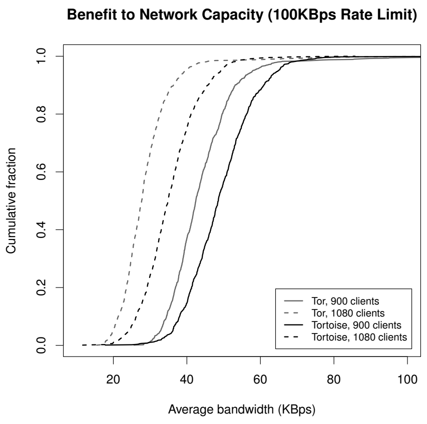


Figure 7: Average bandwidth rates when additional relays join the network when a 100KBps (left) or 200KBps (right) universal rate limit is imposed.



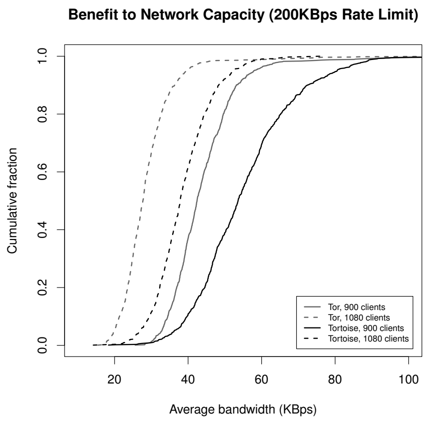


Figure 8: Increased capacity with 100KBps (left) and 200KBps (right) universal rate limits.

nerable to Sybil-style [11] attacks in which a client achieves high bandwidth by multiplexing connections over many Tor circuits. Although each circuit may be individually rate limited, the combined bandwidth may allow the client to surpass the universal rate limit.

Existing Sybil countermeasures may be applied to mitigate such attacks against Tortoise. In particular, guard relays may require clients to complete periodic *cryptopuzzles* [18] in order to continuously forward their traffic. Solving occasional cryptopuzzles will add only a modest burden to standard clients, but could be very computationally expensive for misbehaving clients that connect through many guard nodes. Here, the goal is not to disallow a client from establishing a large number of connections to the Tor network, but rather to shift the incentives to better motivate compliance with Tortoise's envisioned model. That is, applying periodic cryptopuzzles may make it more cost effective to operate an OR rather than to evade the system's universal rate limit.

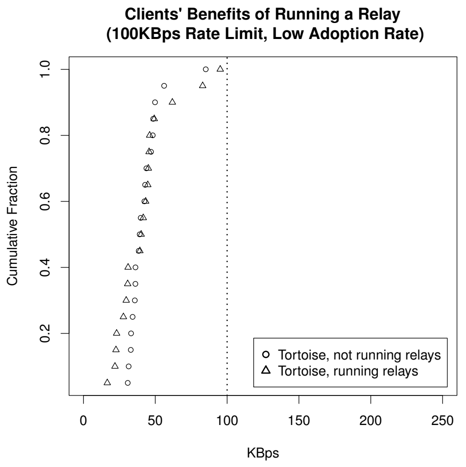
7. RELATED WORK

There are a number of existing approaches that aim to increase Tor's performance. We categorize and outline some of these techniques below.

Prioritizing techniques. Tang and Goldberg recently proposed replacing Tor's round-robin circuit scheduler with one that considers a circuit's recent usage [29]. Using the exponential weighted moving average (EWMA), their technique favors bursty circuits over constantly busy circuits, and consequently lowers the latency of more interactive applications. Their scheduler has been integrated into Tor.

Unlike their approach in which interactive traffic is given precedence over busy clients by rearranging the schedule in favor of the former, Tortoise's traffic shaping is done by active throttling. The two techniques are orthogonal, and EWMA and Tortoise can be simultaneously applied to increase performance.

Improved multiplexing. Reardon and Goldberg note that Tor's TCP multiplexing techniques significantly contribute to network latency. They suggest tunneling TCP connections of DTLS (Datagram Transport Layer Security) packets to alleviate the effects of interference among multiplexed Tor circuits [25]. Similarly, Mathewson has investigated using SCTP (Stream Control Transmission Protocol) for Tor multiplexing [16]. In contrast to these approaches, Tortoise imposes strict rate limits on all non-contributing clients. Applying Tortoise in conjunction with the above multiplexing strate-



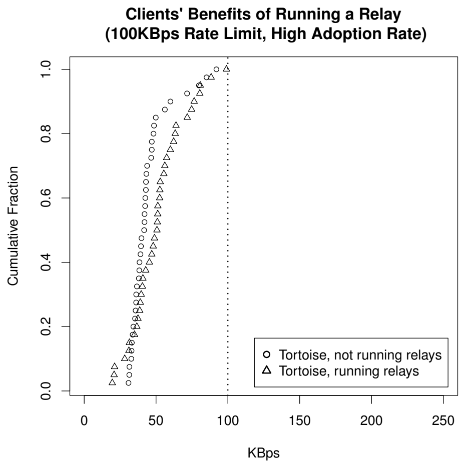


Figure 9: Bandwidth improvements seen by clients that choose to run routers, in a network with only 50% of routers running Tortoise**.**

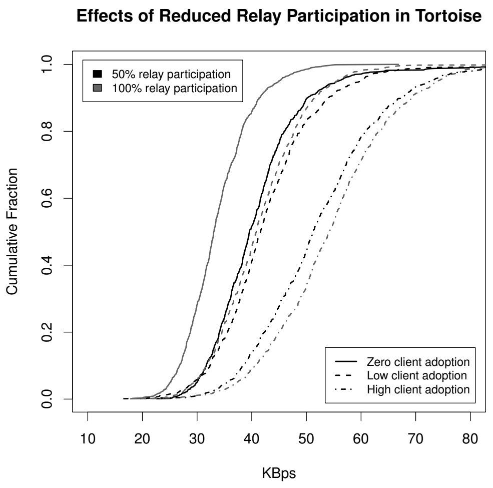


Figure 10: The effect of sub-100% participation in Tortoise **when a 100KBps rate limit is applied.**

gies should provide added benefit, since the reduced traffic load will lighten congestion.

Relay selection. Tor uses a bandwidth-weighted relay selection strategy in which the probability that a relay is chosen as a member of an anonymous path is proportional to the bandwidth advertised by that relay [8]. Snader and Borisov proposed a refinement to Tor that allows the sender to increase the performance of her paths at the expense of anonymity [28]. Murdoch at Watson later showed that Tor's current bandwidth-weighted strategy provides both good anonymity and performance [20]. Sherr *et al.* argue that path selection based on other non-bandwidth metrics (such as latency or loss) offers performance and anonymity benefits [26, 27].

However, as has been pointed out by Dingledine and Murdoch [10], the primary cause of Tor's slowness is likely due to congestion caused by file sharing. While the above relay selection techniques may provide better performance or stronger anonymity on less congested networks, we believe such approaches are unlikely to provide good performance on the current Tor network. Tortoise is agnostic to clients' relay selection algorithms, and may complement the above approaches by alleviating congestion and permitting higher performing and more flexible anonymous routes.

Incentive schemes. Tortoise is most similar to techniques that attempt to provide incentives to operate Tor relays. In PAR [2], relays earn coins that they can spend on fast paths. However, the need to frequently authenticate coins with a central bank limits the approach's scalability.

Jansen *et al.* introduce a number of mechanisms called BRAIDS that encourage Tor users to run relays [14]. BRAIDS implements a form of differentiated service by segregating traffic into three service classes, each of which has particular performance properties (e.g., high latency, high throughput, etc.). Similar to e-cash systems, BRAIDS uses a *ticket* model in which tickets may be exchanged for higher performing anonymous paths. By rewarding relay operators with tickets (and consequently, better performing paths), BRAIDS encourages users to run Tor relays.

Similar to BRAIDS, Tortoise offers a form of differentiated service by enforcing different rate limits for users depending on whether they run a Tor router or not, providing an incentive for users to operate relays. However, while BRAIDS requires a partially trusted offline bank to manage tickets, Tortoise is fully backwards compatible with the existing Tor network, can be incrementally deployed, and requires no centralized structures.

Ngan *et al.* proposed a system [22] in which cells from circuits are marked with a "gold star" if they originate from a Tor instance that is also a STABLE router. Tests on an experimental Tor network showed that using the system, cooperating nodes (nodes running a router, and thus receiving priority) gained a significant advantage over other nodes when the network was under heavy load. While the gold star system considers a cell's priority at each hop, Tortoise checks only at network ingress points, and gives all traffic equal priority once it is past the first hop. Additionally, the ability to prioritize traffic at each hop in the gold star scheme requires that each hop be running the modified software; in Tortoise, only the guard nodes' are aware of the rate limiting.

Adaptive throttling of Tor clients by entry guards. A September 2010 Tor blog post [6] addressed the efficacy of using per-connection rate limits to reduce the impact of bandwidth-intensive connections on the network. The author confirmed that one could set rate limits in a manner such that only large transfers would be significantly throttled. However, some commenters claimed such measures would not help the network because a client could avoid rate limiting by running a relay from the same IP as the client transferring large amounts of data. Our work takes the opposite viewpoint: a bulk client running a relay in order to achieve faster speeds is not only acceptable, it is also desirable as it benefits the network as a whole.

8. CONCLUSION

This paper proposes Tortoise, a backwards-compatible extension to Tor that applies per-connection rate limits at Tor's ingress points. By carefully tuning these limits, our results indicate that Tortoise imposes little performance penalty to most web clients while simultaneously providing incentives to high-bandwidth clients to operate their own relays.

Tortoise's benefits hinge on the ability to attract additional relays. By enforcing strict rate limits and giving exemptions only to relay operators, we argue that clients who demand high bandwidths will be sufficiently motivated to contribute a fraction of their bandwidth to the Tor network. Emulation results demonstrate that even if a small percentage of clients opt to run relays, the network not only achieves significant performance gains, but also an increased capacity to handle additional load. For instance, if 4% of the clients are motivated to operate a relay, then the network experiences a 32% improvement in effective capacity and is significantly better able to tolerate a sudden influx of additional clients than the current Tor network.

Acknowledgements

We thank the anonymous reviewers for their helpful feedback, and Kevin Bauer for several thoughtful discussions about this work. This research was supported by DARPA SAFER award N66001-11-C-4020 and NSF grant CNS-1064986. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the funding agencies.

References

[1] Alexa: The Web Information Company. Top Sites. `http://www.alexa.com/topsites`. Retrieved May 13, 2011.

[2] E. Androulaki, M. Raykova, S. Srivatsan, A. Stavrou, and S. Bellovin. PAR: Payment for Anonymous Routing. In *Privacy Enhancing Technologies Symposium (PETS)*, 2008.

[3] K. Bauer, M. Sherr, D. McCoy, and D. Grunwald. ExperimenTor: A Testbed for Safe and Realistic Tor Experimentation. In *USENIX Workshop on Cyber Security Experimentation and Test (CSET)*, 2011.

[4] S. L. Blond, P. Manils, A. Chaabane, M. A. Kaafar, A. Legout, C. Castellucia, and W. Dabbous. De-anonymizing BitTorrent Users on Tor (poster). In *USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2010.

[5] Y. Chen, R. Sion, and B. Carbunar. XPay: Practical Anonymous Payments for Tor Routing and Other Networked Services. In *ACM Workshop on Privacy in the Electronic Society (WPES)*, 2009.

[6] R. Dingledine. Research Problem: Adaptive Throttling of Tor Clients by Entry Guards. `http://preview.tinyurl.com/3tcyaem`. Retrieved May 24, 2011.

[7] R. Dingledine. Bittorrent Over Tor Isn't a Good Idea. `https://blog.torproject.org/blog/bittorrent-over-tor-isnt-good-idea`, April 2010.

[8] R. Dingledine and N. Mathewson. Tor Path Specification. `http://www.torproject.org/svn/trunk/doc/spec/path-spec.txt`, January 2008.

[9] R. Dingledine, N. Mathewson, and P. Syverson. Tor: The Second-Generation Onion Router. In *USENIX Security Symposium (USENIX)*, 2004.

[10] R. Dingledine and S. Murdoch. Performance Improvements on Tor, or, Why Tor is Slow and What We're Going to Do About It. `https://svn.torproject.org/svn/projects/roadmaps/2009-03-11-performance.pdf`, March 2009.

[11] J. R. Douceur. The Sybil Attack. In *International Workshop on Peer-to-Peer Systems (IPTPS)*, 2002.

[12] F. Hernández-Campos, K. Jeffay, and F. Smith. Tracking the Evolution of Web Traffic: 1995-2003. In *Modeling, Analysis and Simulation of Computer Telecommunications Systems (MASCOTS)*, 2003.

[13] A. Hintz. Fingerprinting Websites Using Traffic Analysis. In *Privacy Enhancing Technologies Symposium (PETS)*, 2003.

[14] R. Jansen, N. Hopper, and Y. Kim. Recruiting New Tor Relays with BRAIDS. In *ACM Conference on Computer and Communications Security (CCS)*, 2010.

[15] K. Loesing. Measuring the Tor Network: Evaluation of Client Requests to the Directories. Technical report, Tor Project, June 2009.

[16] N. Mathewson. Evaluating SCTP for Tor. `http://archives.seul.org/or/dev/Sep-2004/msg00002.html`, September 2004. Listserv posting.

[17] D. McCoy, K. Bauer, D. Grunwald, T. Kohno, and D. Sicker. Shining Light in Dark Places: Understanding the Tor Network. In *Privacy Enhancing Technologies Symposium (PETS)*, 2008.

[18] R. C. Merkle. Secure Communications over Insecure Channels. *Communications of the ACM*, 21:294–299, April 1978.

[19] S. J. Murdoch. Hot or Not: Revealing Hidden Services by Their Clock Skew. In *ACM Conference on Computer and Communications Security (CCS)*, 2006.

[20] S. J. Murdoch and R. N. M. Watson. Metrics for Security and Performance in Low-Latency Anonymity Systems. In *Privacy Enhancing Technologies Symposium (PETS)*, 2008.

[21] NetIndex Source Data. `http://netindex.com/source-data/`. Retrieved May 26, 2011.

[22] T.-W. J. Ngan, R. Dingledine, and D. Wallach. Building Incentives into Tor. In *Financial Cryptography and Data Security*, 2010.

[23] M. Perry. Tips for running an exit node with minimal harassment. `https://blog.torproject.org/blog/tips-running-exit-node-minimal-harassment`. Retrieved May 16, 2011.

[24] M. Perry. Computing Bandwidth Adjustments. Proposal 161, Tor Project, 2009.

[25] J. Reardon and I. Goldberg. Improving Tor using a TCP-over-DTLS Tunnel. In *USENIX Security Symposium (USENIX)*, 2009.

[26] M. Sherr, M. Blaze, and B. T. Loo. Scalable Link-Based Relay Selection for Anonymous Routing. In *Privacy Enhancing Technologies Symposium (PETS)*, August 2009.

[27] M. Sherr, A. Mao, W. R. Marczak, W. Zhou, B. T. Loo, and M. Blaze. A3: An Extensible Platform for Application-Aware Anonymity. In *Network and Distributed System Security Symposium (NDSS)*, 2010.

[28] R. Snader and N. Borisov. A Tune-up for Tor: Improving Security and Performance in the Tor Network. In *Network and Distributed System Security Symposium (NDSS)*, 2008.

[29] C. Tang and I. Goldberg. An Improved Algorithm for Tor Circuit Scheduling. In *ACM Conference on Computer and Communications Security (CCS)*, 2010.

[30] Tor Project, Inc. Tor Directory Protocol, Version 3, 2010. `https://git.torproject.org/checkout/tor/master/doc/spec/dir-spec.txt`.

[31] A. Vahdat, K. Yocum, K. Walsh, P. Mahadevan, D. Kostić, J. Chase, and D. Becker. Scalability and Accuracy in a Large-scale Network Emulator. *SIGOPS Oper. Syst. Rev.*, 36:271–284, December 2002.

[32] S. Zander and S. J. Murdoch. An Improved Clock-Skew Measurement Technique for Revealing Hidden Services. In *USENIX Security Symposium (USENIX)*, 2008.

[33] Zona Publishing. The Need for Speed II. *Zona Market Bulletin*, 5, April 2001.

"Super Nodes" in Tor: Existence and Security Implication

*ChenglongLi[1,3], Yibo Xue[1,2], Yingfei Dong[4] and Dongsheng Wang[1,2]
[1]Tsinghua National Laboratory for Information Science and Technology (TNList), Beijing, 100084, China.
[2]Research Institute of Information Technology (RIIT), Tsinghua University, Beijing, 100084, China.
[3]Department of Computer Science & Technology, Tsinghua University, Beijing, 100084, China.
[4]Department of Electrical Engineering, University of Hawaii, Honolulu, HI 96822, USA.

ABSTRACT

Tor (the second generation onion routing) is arguably the most popular low-latency anonymous communication system now. In this paper, we reexamine the anonymity of Tor based on our observation of "super nodes". These nodes are more available and reliable than other nodes and provide high bandwidth for assisting the system in both performance and stability. We first confirm their existence by analyzing the life cycles of node IP addresses and node bandwidth contributions via two correlation approaches, on a set of self-collected data and a set of real data from the Tor official collection. We then analyze the effect of super nodes on the anonymity of Tor, discuss attacks that exploit such knowledge, and verify our analysis with real data to show potential damages. Furthermore, we investigate new attacks that exploit the knowledge of super nodes. Our simulation results show that these attacks can greatly damage the anonymity of Tor.

Categories and Subject Descriptors

D.6.5 [**Security and Protection**]

General Terms

Security

Keywords

Anonymous communication, Anonymity, Tor

1. INTRODUCTION

Tor is the realization of the second generation onion routing for low-latency anonymous communications. Three types of nodes exist in the Tor network: *directory servers*, *relays* and *bridge relays*. Directory servers are controlled by independent groups[3] and provide global information of Tor relays. Relays are run by volunteers, and they are listed at directory servers and used to build paths for anonymous communications. Bridge relays are not listed in directory servers, and are used as dynamic entry nodes to get around the blocking of public relays by some ISPs. They are chosen by directory servers according to mean uptime and bandwidth[6]. Users learn bridge relays via special emails or sites. The main difference between relays and bridge relays is publically listed or not, and they are both used for building anonymous communication paths. As a result, we consider them the same in our anonymity analysis. For ease of discussion, we use the terms of 'relay' and 'node' alternatively in the following.

According to the Tor official report[4], the daily average number of relay nodes is over 2000, and the daily maximum number of directly connecting Tor users is more than 25000 in February 2011. Tor has a broad user base, including various governments, military agencies, enterprises, and individuals. They mainly use Tor for anonymity. Despite its popularity, Tor still has security issues, and several effective attacks have been developed to compromise its anonymity[18-21]. Tor is sometime mis-used for confidentiality, e.g., the founder of Wikileaks claimed that some documents were intercepted at malicious Tor exit nodes[11, 15].

As Tor becomes more popular in recent years, many volunteers make their relays online as long as possible and contribute more bandwidth. As a result, these nodes contribute more in both performance and reliability. We consider these relay/bridge nodes as "*super nodes*", different from other common users who are less reliable and contribute less bandwidth. In this paper, we investigate how these super nodes affect the anonymity of Tor.

To the best of our knowledge, the existence of super nodes in Tor has not been confirmed before. In this paper, we first verify the existence of super nodes by observing the life cycles of node IP addresses and node bandwidth contributions. Our intuition is that super nodes tend to be more available and more likely to be chosen in building communication paths due to their high bandwidth contribution. However, we found that some of these nodes may associated with different IP addresses over time, due to:(1) some (dial-up) users do not have static IP addresses, e.g., a cable modem user may get a slightly different IP address after rebooting; (2) users may change their IP addresses to avoid being detected and blocked; (3) the network condition of some users may change over time.

To deal with the slight changes of node IP addresses, we use two correlation methods (based on C-Class IP address and BGP prefix) to find the IP addresses that are more likely related to certain relay nodes: long lived in a narrow address range and with high bandwidth contribution. Based on experimental results on our self-collected dataset and a Tor official dataset, we have confirmed that super nodes do exist. While they help the system in both performance and reliability, they may cause new security issues. We will analyze these issues in this paper. The main contributions of this paper are:

- We have confirmed the existence of super nodes in Tor based on experimental data.

ACSAC'11 Dec. 5-9, 2011, Orlando, Florida USA
Copyright 2011 ACM 978-1-4503-0672-0/11/12 …$10.00.

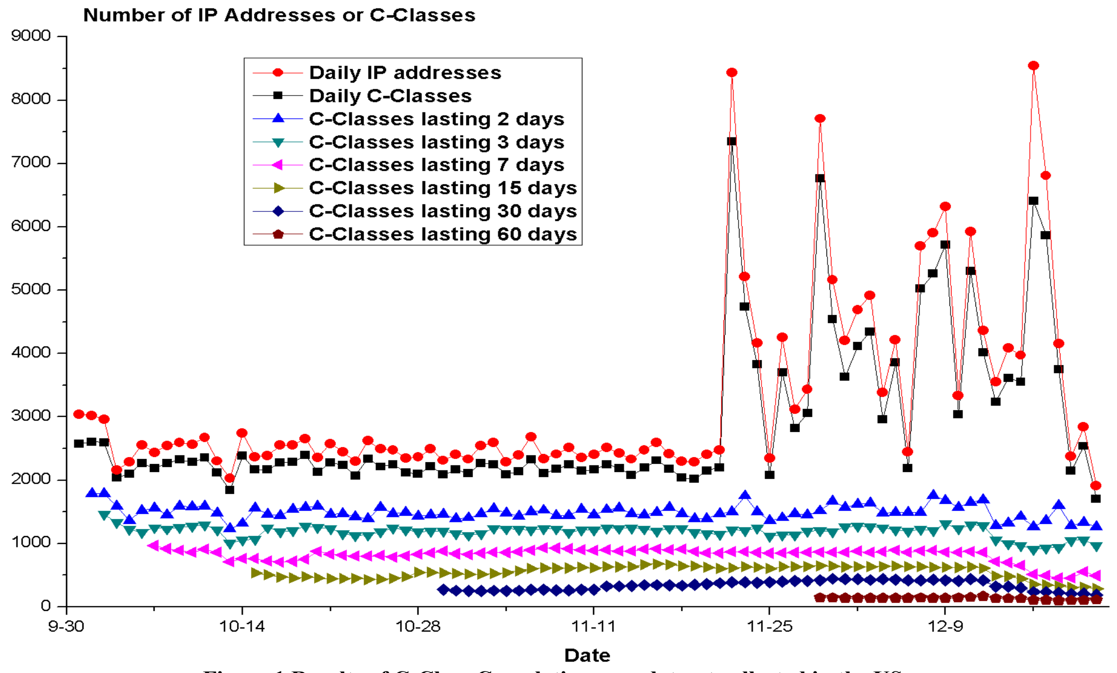


Figure 1.Results of C-Class Correlation on a dataset collected in the US.

- We have analyzed the effect of super nodes on anonymity measure, proposed a new anonymity analysis by considering the effect of super nodes, and discussed the related issues.
- We have investigated potential attacks that exploit the knowledge of super nodes to improve the effectiveness of existing attacks.
- In particular, we have proposed a new theoretical attack which exploits super nodes to cause serious damages.

The remainder of the paper is organized as follows. We will discuss the related work in Sec.2, and introduce the evidences of super nodes in Sec.3. We will then present a new anonymity analysis and discuss its implication in Sec.4. We will discuss several attacks that exploit the knowledge of super nodes in Sec.5, and conclude the paper and discuss our future work in Sec.6.

2. RELATED WORK

Anonymity is one of critical challenges in security systems. Anonymous communication was mostly a pure research topic until 1996: the onion routing[22] was deployed, which is a low-latency anonymous communication system based on Chaum's mix cascades[14]. Tor is the second generation of onion routing, deployed in 2002[3].

The most widely adopted anonymity measure for anonymous communications is the information entropy method [7, 8]. First, entropy is calculated based on the size of effective anonymity set [7], and then the entropy is normalized between 0 and 1 based on the maximum entropy given by the system [8]. Furthermore, the measure of anonymity is defined as a probability $(1-p)$ [9], where p is the probability that a sender is identified by an adversary. Moreover, the degree of anonymity is assessed as $A = \log_2 N$ [10], where N is the number of users in an anonymous network. However, the anonymity measures in [9, 10] cannot appropriately represent anonymity in complex anonymous systems. Therefore, a specification framework for information hiding properties was proposed [12] based on the concept of function view: a concise representation of the attacker's partial knowledge about a function. It describes system behavior as a set of functions, and formalizes different information hiding properties in terms of the views of these functions. Hordes anonymous communication protocol provides similar anonymity as Crowds [9] or onion routing, but with some advantages such as making use of anonymity inherent in multicast routing [13]. Recently, a structural highly-distributed anonymous communication system was proposed to handle scalability issues of Tor and other peer-to-peer systems[2]. Attacks to Tor bridges were proposed, and a membership-concealing overlay network was designed [1]. However, all these existing schemes have not considered the effect of "super nodes" and related security issues in Tor or other anonymous communication systems. In the following, we will first confirm the existence of super nodes, analyze their effect on anonymity, and investigate potential attacks exploiting them.

3. FINDING SUPER NODES IN TOR

By the nature of super nodes, they provide high bandwidth and are long-lived, acting as the "backbone" of Tor for reliability and performance. The Tor path selection algorithm determines their existence. For example, when selecting relays to build a path, it requires that the first 16 bits of the IP addresses of relays are different. As a result, the relays on a path are more likely scattered in different domains. This motivates us to classify relays based on their IP address classes. We then extend the classification with BGP prefixes for more generic cases on the Internet.

We use two correlation methods to analyze the life cycles of C-Class addresses and BGP prefixes of relays, in order to identify the existence of super nodes. The main arguments behind our methods are as follows.

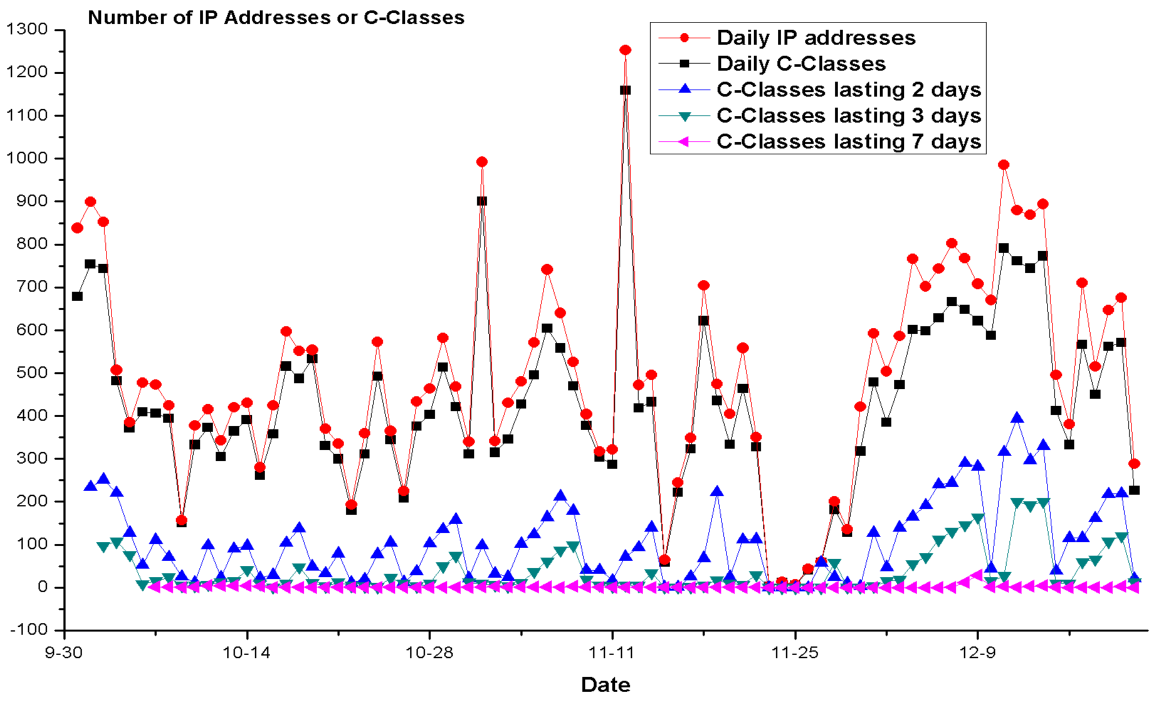


Figure 2.Results of C-Class Correlation in a self-collect dataset in China.

- *One C-Class address is most likely corresponding to one relay node.* Our data reported in the following shows that relays are scattered fairly evenly into different C Classes. (The path selection algorithm builds a path with relays from different C Classes.) The total number of daily IP addresses that we see in the data is very close to the total number of C-Class addresses, i.e., each C Class is likely corresponding to only one relay node.
- *The IP address of a super node is limited to the same C-Class address or BGP prefix.* Although the IP address of a relay may be changed over time, the range of such change is fairly limited in the current IPv4 setting, mostly within the same C Class address or BGP prefix.
- *Tor needs to have a certain number of stable nodes to achieve a proper level of anonymity.* As any anonymous communication systems, the total number of nodes in the system determines the anonymity measure. We try to find the set of reliable nodes by analyzing the life cycles of node IP addresses over different time intervals.
- *Tor path selection algorithm somehow determines which types of nodes become super nodes,* because the bandwidth and flags (including capacity and uptime) are the main factors in the current path selection algorithm. Our analysis in the following confirms that super nodes are *long-lived* and usually have *high bandwidth contribution*.
- *We can easily filter out most normal relays by examining their life cycles and bandwidth contributions.* A normal volunteer usually runs a Tor relay with limited resources for relative short periods and leaves based on its own needs. Because it is less reliable and less capable, it is less likely to be selected in a Tor path. Its IP address (and its associated C-Class address/BGP prefix) is only shown up shortly in the data set.

We obtained two datasets: one is self-collected, and the other is from the Tor official collection.

- Self-collected Datasets: We collected the IP addresses, bandwidth, and other information of Tor relay nodes in both US and China from Sept. 30th, 2010 to Dec. 21st, 2010. In China, we run a Tor relay node without exit permission in the CERNET (Chinese education network), and collected information of Tor nodes connected to it. In the US, we use the same method on a VPS (Virtual Private Server) of *Linode*[16].
- Official Dataset: This dataset is acquired from the Tor official metrics site [4]. We analyzed official consensus data on the site from Jan. 1st, 2010 to Dec. 31st, 2010.

3.1 C-Class Correlation

As supper nodes tend to be online for long periods, we examine the life cycles of C Classes of the *daily* relay IP addresses in each dataset over continuous time intervals of **i** consecutive days, where **i**= 2, 3, 7, 15, 30 or 60.(It represents the uptime of C Classes.) The results of the self-collected data in the US and China are shown in Fig.1 and Fig.2, respectively.

First, the *daily* number of C Classes of relays is just a little smaller than that of *daily* IP addresses of relays in both figures. This means that the IP addresses of relays are fairly evenly distributed among C Classes, both in the US and China. In other words, in most cases, one relay IP address is corresponding to one C-Class address.

Second, as we increase the interval length, the number of *persistent* C Classes in Tor changes very little. By *persistent*, we mean that these C Classes show up every day during the interval. As we see, the large daily fluctuation of normal-relay joins/leaves does not affect the Tor "backbone", consisting of long-lived nodes. In our self-collected datasets, we have seen x=2824 *daily* C Classes occupied by Tor nodes. Without loss of generality, we

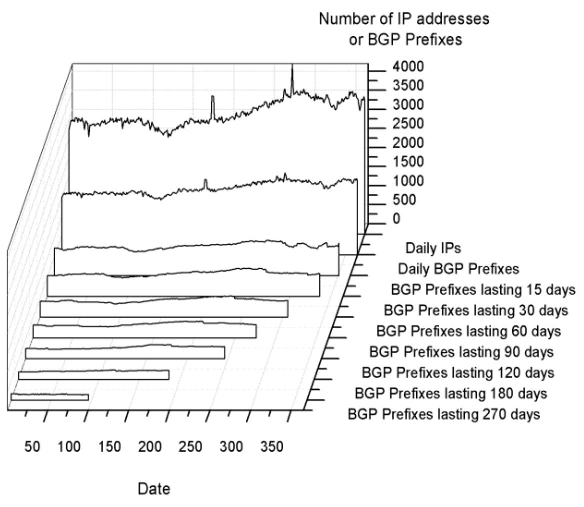


Figure 3.Results of BGP Prefix Correlation.

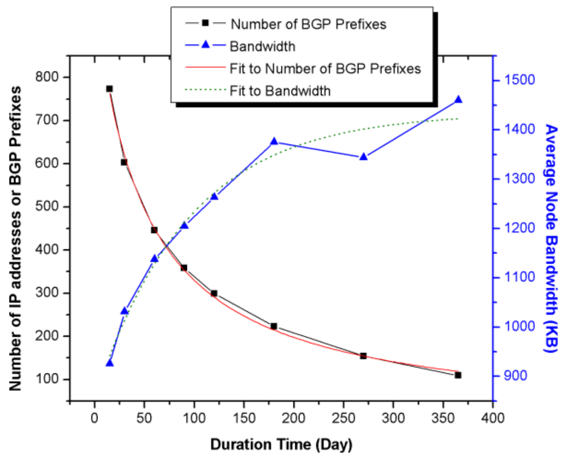


Figure 4. Average Prefix Number and Bandwidth of super nodes.

assume that a node join is independent to the current node distribution. For a node join, the probability that the new node belongs to the existing $\boldsymbol{x}$ persistent C Classes is very small ($\boldsymbol{x}/\boldsymbol{y} \approx 0.02\%$), where $\boldsymbol{y}$ is the total number of C Classes on the Internet, and $\boldsymbol{y}=2^{24}$. The main reason behind the small change of the number of C Classes is that an existing node likely changes its IP address in the same C Class, or the same Classless Inter-domain Routing (CRID) block, corresponding to a BGP prefix.
Third, in Fig.1 (*data collected in the US*), the number of *daily IP addresses* (the top curve) is quite smooth from the beginning to Nov. 21st (with some fluctuations for the remainder of the period). The number of *daily C Classes* (the second curve from the top) behaves similarly. However, the number of persistent C Classes over several days (the six curves at the bottom) is relatively steady all the time for all cases. This fact indicates that the basic structure of Tor network is mainly stable without dramatic changes in spite of the fluctuation of daily relay IP addresses. About $\frac{1}{3}$ of C Classes are still active after seven days. Over 100 C Classes are active more than60consecutive days during the 83-day data collection period. We believe that these nodes are likely the foundation of Tor, supporting its basic anonymity and service quality. They are very likely the “super nodes” that we are looking for. In Fig.2 (*data collected in China*), the results are quite different due to network censorship in China. The number of *persistent* C Classes over several days (the bottom three curves) is very low, compared with that in the US. The number of persistent C Classes drops very fast and often reaches zero after seven days. It is most likely due to active filtering and blocking.

In summary, the data collected in the US reflects common cases on the Internet. The number of persistent C Classes tells us that super nodes are quite stable in spite of a large number of normal nodes join/leave. The number of C Classes over seven (or more) days has almost no change, i.e., super nodes are long lived.

3.2 BGP Prefix Correlation

To further verify the existence of super nodes, we also analyze the BGP-prefixes associated with relays in the Tor official dataset. We chose one-year official consensus data and calculated the daily relay IP addresses in the same BGP prefixes over continuous time intervals of **i** consecutive days, where **i**= 15, 30, 60, 90, 120, 180, 270, and 365. The BGP table was obtained from [5] in Jan., 2011. Fig.3 and Fig.4 show the results. More than 100 *persistent* BGP prefixes were active across the whole year as shown in Fig.3. The number of persistent prefixes (the left Y-axis in Fig.4) drops slowly as we increase the interval length. (One year may be too long to be justified as a super node, because it may switch to another ISP or replaced by a new node for various reasons. We just use it as an example to show the behavior.)

There were$\boldsymbol{y'} \approx 344{,}000$ different BGP prefixes in the BGP table [5]. The peak number of *daily* BGP prefixes of Tor relays is $\boldsymbol{x'} = 2152$. Assuming Tor nodes have no social relations in the real world, a new relay IP address appears in the data due to two cases: a new node joins, or an existing node changes its address. Our results show that a Tor node is likely distributed in a unique (BGP-prefix) address space, while a new node join is only adding a new *daily* BGP prefix into the dataset. The probability that the new address belongs to $\boldsymbol{x'}$existing BGP prefixes is independent to the current node distribution, which equals to $\boldsymbol{x'}/\boldsymbol{y'} \approx 0.06\%$. When an existing relay changes its IP address, it is likely to under the same BGP prefix. The results are similar to those seen in the above C-Class address analysis.

The number of super nodes we see depends on how we define the correlation interval. As shown in Fig.4, it varies from about one thousand to several hundred on the current Tor based on different views. We also calculate the average node bandwidth (the right Y-axis in Fig.4) associated with these persistent BGP prefixes in different time intervals. We firstly calculate the average bandwidth of nodes belonging to the same prefix respectively, and then obtain the final average bandwidth of all persistent BGP prefixes in an interval. As shown in Fig. 4, the average bandwidth of these nodes increases with the interval length. The number of persistent prefixes fits the reciprocal function $\mathrm{y} = 1/(\mathrm{a} + \mathrm{b} * \mathrm{x})$ well, where a=0.001 represents the adjusting parameter, and b=0.00002 represents the attenuation factor. The average bandwidth of these nodes fits the exponential function $\mathrm{y} = \mathrm{y}_0 + \mathrm{A}^{\mathrm{r}*\mathrm{x}}$ well, where y_0=1435.06, A=-578.27, and r=-0.0106.

3.3 More Discussions of Super Nodes

The Tor path selection algorithm clearly separates super nodes from normal nodes based on reliability and bandwidth (or throughput) contribution. Combining these two parameters together, we can determine the set of super nodes. From the Tor official metrics data [4], the advertised average daily node

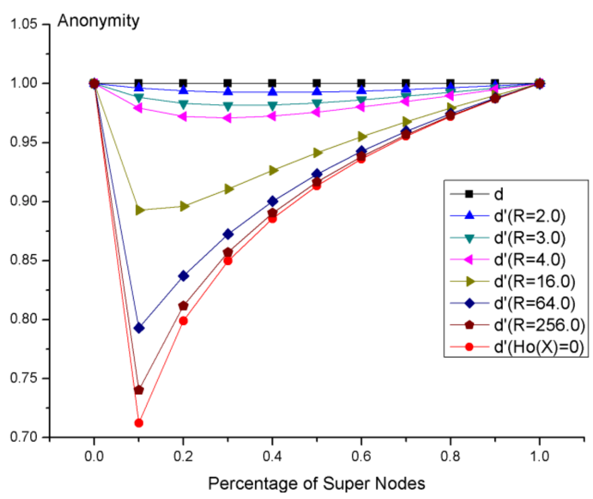


Figure 5.Comparison of d and d′.

bandwidth in 2010 is 354 KB; the real traffic read-and-written by a relay (recorded from April 29th, 2010) has a daily average of 264 KB. As the latest Tor path selection algorithm considers node reliability and bandwidth as critical parameters, it is clear that the uptime and the bandwidth of super nodes are very consistent in the data. The nodes with a long life cycle (even changing their IP addresses slightly) provide more bandwidth for anonymous communications. Because Tor tends to select nodes with more bandwidth, we define the *selection ratio of super nodes over normal nodes* as R, which is the average bandwidth of super nodes over the average bandwidth of normal nodes. It is about 2.6 based on the 15-day correlation interval and about 4 based on the one-year correlation interval. It becomes larger as the interval length increases. Clearly super nodes have significant influence on both system stability and performance.

In summary, we have used two correlation methods to confirm the existence of super nodes over a self-collected data and the Tor official data. We have also identified that the long-lived super nodes usually provide higher bandwidth. We consider these results significant in affecting the anonymity in Tor, because (i) Identifying super nodes from common relay nodes affects the anonymity metrics. (ii) Attackers can exploit the difference to improve existing attacks and introduce new attacks exploiting super nodes and their characteristics. As a result, the defense mechanism of Tor may have to be improved correspondingly. In the next section, we will examine how anonymity is affected due to the existence of super nodes in Tor.

4. THEORETICAL ANALYSIS

4.1 Simple Anonymity Formulation

When using information theory to define anonymity[7, 8], an anonymity set is denoted as$\mathrm{A} = \{\mathrm{a}_1, \mathrm{a}_2, \dots, \mathrm{a}_n\}$, and $\mathrm{n} = |\mathrm{A}|$.$X = \{\mathrm{x}_1, \mathrm{x}_2, \dots, \mathrm{x}_n\}$isa discrete random variable and its probability density is denoted as $p_i = \mathrm{P}(\mathrm{x}_i)$, and $\sum_{i=1}^{n} p_i = 1$. Thus the entropy of X is defined as $\mathrm{H}(X) = -\sum_{i=1}^{n} p_i \log_2(p_i)$. Assume $\mathrm{H_M}$is the maximum entropy, and $\mathrm{H_M} = \log_2(\mathrm{n})$. The information that an attacker can acquire is represented as $\mathrm{H_M} - \mathrm{H(X)}$. So the anonymity of a communication system is defined as

$$\mathrm{d} = 1 - \frac{\mathrm{H_M} - \mathrm{H(X)}}{\mathrm{H_M}} = \frac{\mathrm{H(X)}}{\mathrm{H_M}} \qquad (1)$$

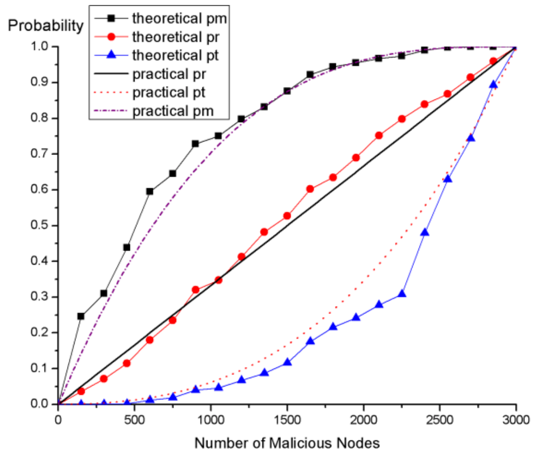


Figure 6.Evaluation of $\mathbf{p_m}$, $\mathbf{p_r}$ and $\mathbf{p_t}$.

Clearly, $0 \le d \le 1$, and d=0 when there is only one element in the anonymity set. (Or, when certain p_i=1, the anonymity of the system is minimal, d=0.) While p_i=1/N, the system anonymity reach its maximum, d=1.

In an ideal Tor, every relay is treated as the same in a Mix-net system. Super nodes and normal nodes act in the same fashion for forwarding packets. However, as we demonstrated in the previous section, because super nodes can be identified by correlation, we have to consider them in a different category when evaluating the system anonymity, i.e., the existence of super nodes affect the anonymity, stability, and usability of Tor.

In the following, we reexamine system anonymity by considering the existence of super nodes. We divide the same anonymity set Aintoa super-node subset $A_u = \{\mathrm{a}_1, \mathrm{a}_2, \dots, \mathrm{a}_t\}$ and a normal-relay subset $A_o = \{\mathrm{a}_{t+1}, \dots, \mathrm{a}_n\}$. We have $\mathrm{n} = |\mathrm{A}|$, $n_u = |A_u|$, $n_o = |A_o|$, and $n = n_u + n_o$. X is a discrete random variable and its probability density is denoted as $p_i = \mathrm{P}(\mathrm{x}_i)$, and $\sum_{i=1}^{t} p_i + \sum_{i=t+1}^{n} p_i = \sum_{i=1}^{n} p_i = 1$.Thus the entropy of X is denoted as $H(X) = -\sum_{i=1}^{n} p_i \log_2(p_i) = -(\sum_{i=1}^{t} p_i \log_2(p_i) + \sum_{i=t+1}^{n} p_i \log_2(p_i)) = H_u(X) + H_o(X)$. Based on formula (1), the anonymity of the system with super nodes is denoted as

$$d' = \frac{H(X)}{H_M} = \frac{H_u(X) + H_o(X)}{H_M} (2)$$

Based on this new definition, without the knowledge of super nodes, the anonymity of Tor network is the same as before. However, when distinguishing super nodes from normal nodes, the system anonymity is reduced.

Assume we have a total of 3000 nodes in the network, similar to the number of daily nodes in the current Tor. Assume under the ideal conditions the same type of nodes has the same probability density, we have d=1. Now, let us consider the probability density of a super node is **R** times higher than that of a normal node where **R** is defined in Sec.3.3. We choose **R** = 2, 3, 4, 8, 16, 64, 256to compare d and d', as shown in Fig.5. Clearly, when nodes are divided into two types, the anonymity of the system is significantly reduced. (i) When the number of super nodes reaches a certain percentage in the system, the anonymity reaches the minimal value. For example, when **R**=4.0 and the percentage of super nodes is 30%, the attacker's chance increase from the ideal 0%to 3%. This means that the attacker has almost no chances before and now it has a good chance if it repeats the attack enough times. (ii) The ratio **R** heavily affects the anonymity. When **R**

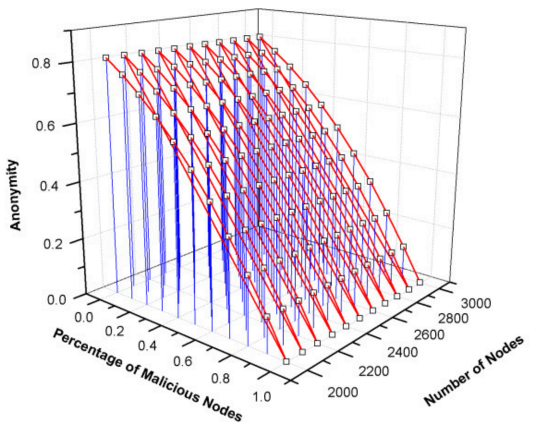


Figure 7.Anonymity under Malicious-Node Attack.

increases (the differences between a super node and a normal node become larger), the attacker will have much higher chances, e.g., from 0% to 10% when **R**=16.

We also consider the worst case in which attackers focus on super nodes only. In this case, the probability density of a normal relay is $p_i = 0$, where node i $\in A_o$, thus $H_o(X) = 0$. As the bottom curve shown in Fig.5, the lowest anonymity entropy reaches 70% (a 30% drop), a dramatic decline of system anonymity. Clearly, the existence of super nodes significantly damages anonymity. In the following, we will further examine anonymity under concrete attack models.

4.2 Different Attack Mode

When considering attacks exploiting the knowledge of super nodes, we examine several different situations as follows.

4.2.1 Brute-Force Attack on Service Availability

A brute-force attack may block (or deny the service of)a set of nodes in the system to make Tor unavailable, maybe initiated by a powerful attacker such as an ISP (or a government).Even with anti-censorship technology in which a user can connect to Tor by (semi-hidden) bridges, blocking (or DoS) super nodes is still a great challenge to the usability and stability of Tor. When focusing on super nodes instead of just any relays, these attacks can be more effective.

Let us first consider system performance. Assume we have ***s*** super nodes in a system of ***n*** nodes. Assume the bandwidth contribution of the whole system is ***B***. Then the overall contribution in a time interval ***T*** is ***C=T*B***. Let us denote the average bandwidth of super nodes as $\boldsymbol{b_1}$, and the average bandwidth of normal nodes as $\boldsymbol{b_2}$. So the contribution of super nodes is $\boldsymbol{C_1=Tb_1}$, and the contribution of normal nodes is $\boldsymbol{C_2=Tb_2}$, and $\boldsymbol{C=C_1+C_2}$. Based on the analysis of super nodes in the Tor official data, the contribution made by super nodes is about 66% of the contribution of the whole system ($\mathrm{C_1/C = 66\%}$), with only 21% of nodes are super nodes ($\mathrm{s/n = 21\%}$).

Let us now consider the system anonymity and security. When the system suffers a brute-force attack without the knowledge of super nodes, the anonymity measure will decrease linearly with the number of nodes being attacked, based on formula (1). However, when attackers recognize super nodes, they aim at super nodes to achieve more damages, because Tor uses a bandwidth-weighted path selection mechanism. In general, the probability of node ***i*** being chosen for building a Tor path is $\mathrm{b_i / \sum_{k=1}^{n} b_k}$, where $\boldsymbol{b_i}$ is the bandwidth contributed of node ***i***. So the influence of super nodes on anonymity is $\boldsymbol{b_1/b_2}$ times than that of normal nodes. The maximum value of this ratio is 4 in the dataset we collected, which means the impact of attacking one super node equals to that of attacking four normal nodes. To maximize attack impact, attackers will aim at super nodes. In this case, $p_i = 0$, when i is a normal node; when i is a normal node, $p_i = \frac{1}{n_u}$, the maximum anonymity measure is obtained as $d_{max} = \frac{H_u(X)+H_o(X)}{H_M} = \frac{H_u(X)}{H_M} = \frac{\log_2 n_u}{\log_2 n}$. Thus, when super nodes are recognized, the system anonymity is reduced.

4.2.2 Malicious-Node Attack

If an attacker controls some nodes in the system, it can launch different kinds of complex attacks, including timing correlation attacks, disclosure attacks, predecessor attacks, collusion attacks, or Sybil attacks. We now focus on a simple malicious-node attack, in which an attacker compromises the first or the last relay node or both ends of a three-hop onion routing path.

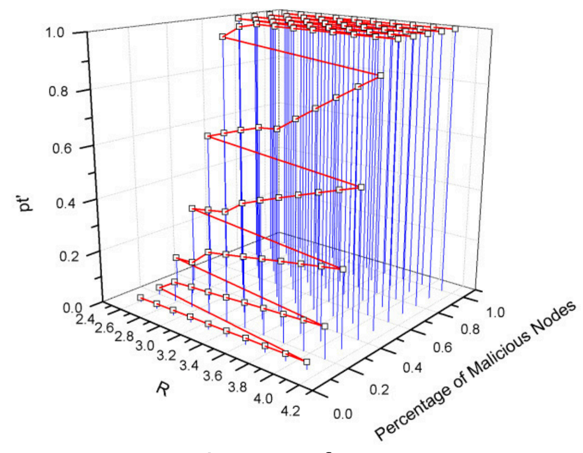
The position of a malicious node on a path is critical in this attack, because a node only knows its predecessor and successor on the path in Tor. When a malicious node is at the first hop of a path, the sender identity, its traffic patterns (e.g., packet size or timing), and other sender information is revealed. When a malicious node is at the last hop (acting as an exit node), the receiver and the plain-text traffic is exposed. When a malicious node is at the middle hop (the default path length is 3), only other relay nodes are revealed, and it will not affect the system security unless combined with other attacks. When an attacker controls both ends of a path, the sender, the receiver, and their relationship, are all exposed. Therefore, different positions on a path should be treated differently when calculating the anonymity measure under this malicious-node attack model.

Assume the system has ***n*** nodes, and ***c*** nodes out of ***n*** nodes are malicious nodes. The default path length ***L*** is three in Tor. The probability that attacker knows the sender is $\mathrm{p_r = c/n}$. The probability that attacker only know the receiver is the same as $\mathrm{p_r}$. As the probability that all three hops are all non-malicious nodes is $\frac{n-c}{n} * \frac{n-c-1}{n-1} * \frac{n-c-2}{n-2}$, we know the probability that a path includes at least a malicious node as $\mathrm{p_m} = 1 - \frac{n-c}{n} * \frac{n-c-1}{n-1} * \frac{n-c-2}{n-2}$. We define the joint probability that the first hop and the last hop of a path are both malicious nodes as $\mathrm{p_t}$. To find $\mathrm{p_t}$, we need consider two cases: only the first and the last hop are malicious nodes and all three hops are malicious nodes. We have $\mathrm{p_t} = \frac{c}{n} * \frac{c-1}{n-1} * \frac{n-c}{n-2} * \frac{P_2^2}{P_3^3} + \frac{c}{n} * \frac{c-1}{n-1} * \frac{c-2}{n-2}$, when malicious nodes are chosen randomly and regardless of the distribution of nodes. We present the theoretical analysis to $\mathrm{p_m}$, $\mathrm{p_r}$ and $\mathrm{p_t}$ in Fig.6. We further generalize the anonymity measure formation for any path length **L** based on information entropy theory as

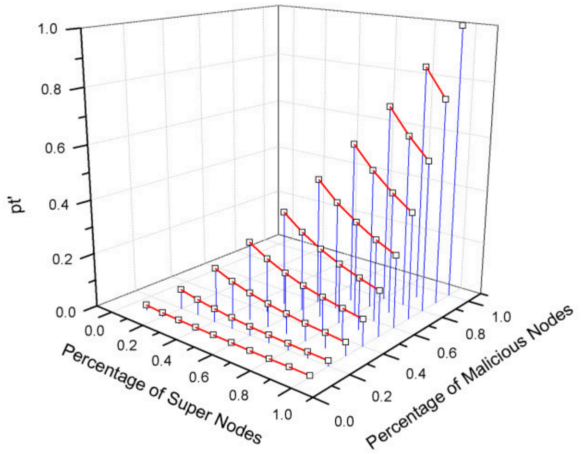
$$\mathrm{d''} = \frac{\mathrm{H(X)}}{\mathrm{H_M}} =$$

$$\frac{\mathrm{n*\sum_{k=2}^{L}\frac{p_{n-c}^{k-1}}{p_n^k}}}{\left(\mathrm{1+n*\sum_{k=2}^{L}\frac{p_{n-c}^{k-1}}{p_n^k}}\right)\mathrm{*\log_2(n-c)}} * \log_2 \frac{\mathrm{(n-c)*}\left(\mathrm{1+n*\sum_{k=2}^{L}\frac{p_{n-c}^{k-1}}{p_n^k}}\right)}{\mathrm{n*\sum_{k=2}^{L}\frac{p_{n-c}^{k-1}}{p_n^k}}} +$$

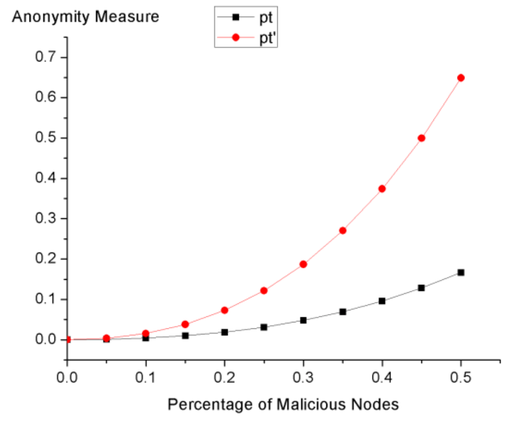
$$\frac{1}{\left(\mathrm{1+n*\sum_{k=2}^{L}\frac{p_{n-c}^{k-1}}{p_n^k}}\right)\mathrm{*\log_2(n-c)}} * \log_2\left(1 + \mathrm{n} * \sum_{\mathrm{k=2}}^{\mathrm{L}} \frac{\mathrm{p_{n-c}^{k-1}}}{\mathrm{p_n^k}}\right) \qquad (3)$$



a.p_t' with fixed $\frac{s}{n} = 0.2$



b.p_t' with fixed $R = 2.61$



c.p_t'vsp_t with fixed R and$\frac{s}{n}$

Figure 8. The analysis to new anonymity measure p_t'.

In formula (3), H_M is the maximum entropy of the system, H(X) is the entropy of stochastic variable X, and $\boldsymbol{k}$ is the hop position where a malicious node appears on the path.

Using formula (3), the anonymity measure is shown in Fig.7. When there is no malicious node, the anonymity is about 0.8. The reason that the anonymity cannot reach 1 is because the default Tor path length is three. When $\boldsymbol{L}$ becomes larger, the anonymity will converge to 1 without malicious nodes. The clear drops in the figure show that the percentage of malicious nodes in the system heavily affects the anonymity measure. The total number of nodes in the system does not affect the anonymity measure very much when the system reaches a certain size. For example, the anonymity measure varies slightly as we increase the total number of nodes from 2000 to 3000 nodes in the tests. In the following, we will further examine the system anonymity under malicious nodes attacks which exploit the existence of super nodes.

5. SECURITY CONCERNS DUE TO THE EXISTENCE OF SUPER NODES

Attackers can discover super nodes as we did, and use the information to compromise the system anonymity by improving known attacks or developing new attacks.

5.1 Improving Known Attacks

With the knowledge of super nodes, the effectiveness of known attacks can be improved. To evaluate such impact, we first analyze its damages in this section and then evaluate it on a simulation platform in the next section.

5.1.1 Brute-Force Attack

When under a brute-force attack, attacking all nodes without differences is less effective because of the large number of relay nodes. However, attackers can focus their attacks on super nodes to damage the stability and usability of Tor, or filter all traffic passing through super nodes to reduce the availability of Tor. As mentioned in the above, the average bandwidth contribution of super nodes is about 66% of the whole system with only 21% of nodes. Therefore, attacker can effectively cripple 66% of network traffic by aiming at only 21% nodes. The attack is about three times more effective.

5.1.2 Malicious-Node Attack

Under malicious-node attacks, we use p_m, p_r and p_t to characterize the catalytic role of super nodes. Again, we have total $\boldsymbol{n}$ nodes in the system with $\boldsymbol{s}$ super nodes and $\boldsymbol{c}$ malicious nodes. Assume $\boldsymbol{b_k}$is the bandwidth of node $\boldsymbol{k}$, and$\boldsymbol{b_1}$represents the bandwidth of a super node and $\boldsymbol{b_2}$represents the bandwidth of a normal node. We define the selection factor of super nodes as $\frac{b_1}{\sum_{k=1}^{n} b_k}$, and the selection factor of normal nodes as$\frac{b_2}{\sum_{k=1}^{n} b_k}$. So a super node is $R = \frac{b_1}{b_2}$times likely to be chosen in a path, as the *selection ratio of super nodes over normal nodes* discussed in Sec.3.3. Assuming the original bandwidth of each node is the same equal to 1, when the total bandwidth of the system is B=n, we have $b_1 = \frac{R*n}{(R-1)*s+n}$and $b_2 = \frac{n}{(R-1)*s+n}$. Recall that $p_m = 1 - \frac{n-c}{n} * \frac{n-c-1}{n-1} * \frac{n-c-2}{n-2}$, $p_r = \frac{c}{n}$ and $p_t = \frac{c}{n} * \frac{c-1}{n-1} * \frac{n-c}{n-2} * \frac{P_2^2}{P_3^3} + \frac{c}{n} * \frac{c-1}{n-1} * \frac{c-2}{n-2}$ in the basic model of malicious node attack. Now, when attacks aim at super nodes, we focus on the impact on p_t. Because compromising the both ends of a path damages anonymity the most, we have

$$p_t' = {b_1}^2 * p_t = \left(\frac{R*n}{(R-1)*s+n}\right)^2 * \left(\frac{c}{n} * \frac{c-1}{n-1} * \frac{n-c}{n-2} * \frac{P_2^2}{P_3^3} + \frac{c}{n} * \frac{c-1}{n-1} * \frac{c-2}{n-2}\right).$$

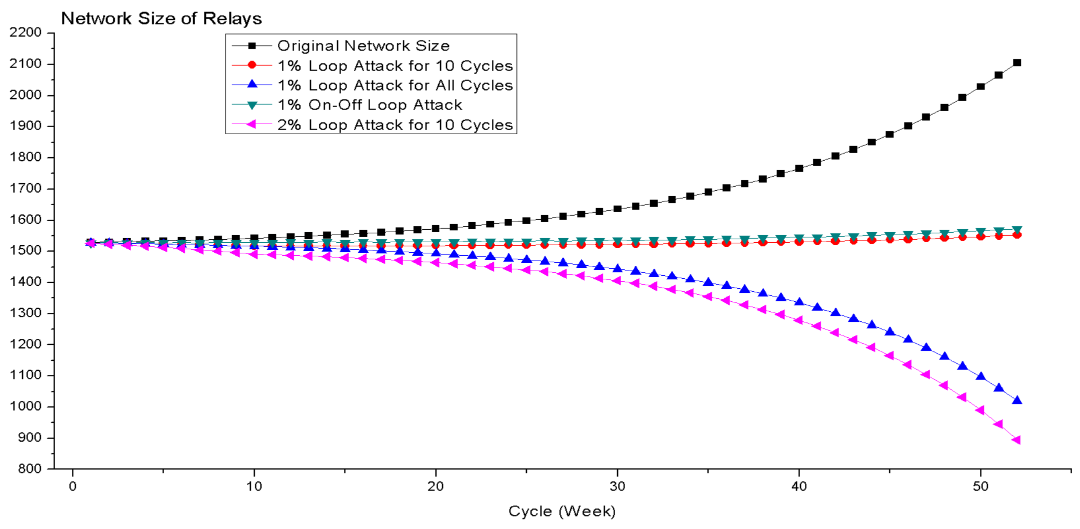


Figure 9.Analysis to Loop Attack.

When we fix the number of super node **s** as 600 in the total of **n**=3000 nodes, the result is shown in Fig.8(a). An attacker will prefer to compromise super nodes than normal nodes. When 900 nodes (30% of the total nodes) are *malicious nodes* (with 600 super nodes among them), the anonymity measure p'_t is about 0.3.

Meanwhile, p'_t becomes higher with the increase of **R**, while the original p_t is only related with the total number of malicious nodes. The range of **R** is between 2.6 to 4 from the real data presented in Sec.3.2. (However, **R** may be larger in many cases.) Here we set **R** to its minimum value of 2.6, and the results is shown in Fig.8(b). There are fewer points shown in the figure because we set the number of malicious nodes less than the number of super nodes to show the effect when *the attack focuses on super nodes only*. Both the percentage of super nodes and the percentage of malicious nodes have significant influences on p'_t. p'_t increases exponentially with the increase of the percentage of malicious nodes.

We further compare p_t and p'_t when $R = 2.6$ and $s/n = 0.5$, as shown in Fig.8(c). When all super nodes are malicious, p'_t increases more rapidly than p_t, and reaches about 0.6, which is over three times more than p_t (at the same percentage of malicious nodes). In summary, the above simple analysis shows that, aiming at super nodes, malicious-node attacks can be improved significantly.

5.1.3 Other Attacks

Besides brute-force attacks and malicious-node attacks, attackers can control some super nodes and configure them as exit nodes. By capturing traffic of these exit super nodes, attackers intercept plain-text traffic, just like what Wikileaks did but more effective.

5.2 Potential New Attack

Here we introduce a new attack method, called *loop attack*, which is designed for attackers managing a large section of a network. This attack can amplify the knowledge of super nodes repeatedly to lower the perceptibility of users who are located in the network managed by the attacker. First, the adversary starts normal attacks to super nodes, e.g., just blocking them. The users may still use the Tor service with poor availability and performance. If the number of affected users is sufficient large, assuming the attacker is a large ISP or a government, the total number of Tor users in the network will decrease. Such a decrease leads to even worse performance, anonymity and availability of Tor, and then further lower the user confidence in the system and thus the number of users. With such a vicious cycle, the total number of users will reduce gradually. The attacker can also launch other attacks at the mean time to enhance the effect of this attack. Normal users are not aware of the attack; Tor operators also have a hard time to detect the attack when the statistics are changed gradually. So this attack is invisible at the beginning until its accumulative damage to some extent.

We model the effect of this attack as follows. The notation is summarized in Table 1. Assume the number of nodes following the simple iteration: $n_{t+1} = n_t + \alpha_t$, $\alpha_t = w_d * (d_t - d_0) + w_c * (c_t - c_0) + \beta$, where α_t can be either positive or negative, representing the increase or decrease of the number of nodes. It is affected by the user experience of Tor in both security and performance. Let d_t denotes the anonymity entropy of the system in cycle t, and c_t denotes the contribution of the system in cycle t. Let d_0 denotes the threshold of anonymity of the system, and c_0 represents the threshold of contribution of the system, which are both calculated based on the initial data of the system. We use w_d and w_c asan anonymity weight and a contribution weight, which are used to balance their influences. The total number of nodes in cycle t is denoted as n_t, in which s_t are super nodes. The average bandwidth of super nodes is $b_1 = R * n_t/((R-1) * s_t + n_t)$, and the average bandwidth of normal nodes is $b_2 = n_t/((R-1) * s_t + n_t)$. Again, attackers prefer super nodes over normal nodes with a ratio $\boldsymbol{R=b_1/b_2}$. Thus the overall contribution of the system in cycle t is $c_t = b_1 * s_t + b_2 * (n_t - s_t)$. The estimated contribution determines system performance and user experience.

To evaluate the anonymity of the system in cycle t, we approximate the entropy as $d_t = \log_2 n_t$ in cycle t. We set the parameters based on the Tor official dataset of Year 2010.We obtain the values which fit the data well. We choose one loop iteration as one cycle, and consider one cycle as a week. There are 52 cycles in the data.

The effect of the new attack is shown in Fig. 9. We consider several different stealth attack strategies. (1) *1% loop attack* means that 1% of all nodes and 5% of super nodes are blocked by the attacker. By blocking the same number of nodes over only 10

Table 1.Parameters of Loop Attack

Parameter	Implication	Value
n_t	Network size in cycle t	-
α_t	Change of network size in cycle t	-
w_d	Weight of anonymity	40
d_t	Entropy in cycle t	-
d_0	Entropy threshold	10.5774
w_c	Weight of Contribution	0.04
c_t	Contribution in cycle t	-
c_0	Contribution threshold	1528
β	Const	1
R	Selection Ratio	2.6
s_t	Number of super nodes in cycle t	n_t*20%

cycles, the network size increases very little in one year (shown as the third curve from the top), compared to the original increase of several hundred nodes (the top curve). (2) *1% on-off attack* means that the attacker blocks 1% of all nodes and 5% super nodes in one cycle, and unblocks them in the next cycle, and so on. Although the results of 1% on-off attack (the second curve from the top) are similar to that of 1% loop attack over 10 cycles, the on-off alternative nature makes it hard to detect. (3) *1% attack over all cycles* means that the attacker blocks 1% nodes and 5% of super nodes over all the cycles. (4) *2% attack over 10 cycles* means that the attacker blocks 2% of nodes and 10% of super nodes. Clearly, case 3 and case 4 (the bottom two curves) cause the number of users drop significantly in 52 cycles. As we see, the loop attack on super nodes generates a vicious cycle in the system.

The main feature of this new attack is to utilize the knowledge of super nodes. The advantage of this attack is its concealment. Attackers only need to attack 1% or 2% of nodes in the system based on the knowledge of super nodes. Consequently the network size is reduced significantly, which serious damages the anonymity measure and performance of the system. Note that we use the minimum value of R=2.6 from our data collection. R could be larger in other cases, where the effect of this attack becomes more significant.

6. EXPERIMENT EVALUATION

6.1 Simulation Platform

We have evaluated our analysis with experiments set up in our lab. Because it is impossible to test our methods on the real Tor, we have built a simulation platform to simulate the entire Tor system, in order to evaluate the practical implication of super nodes on performance and anonymity. We simulate Tor algorithms with 3000 nodes. A node in the simulation platform behaves as the exactly same as a node in the real Tor. The only difference is that we have the complete control of each node to set up attacks and collect data. Besides relay nodes, directory servers are also simulated. The simulation platform can perform node joins, path selection, various attacks, etc. To the best of our knowledge, this is the first large-scale simulation platform of Tor, which has the same size as the current Tor. It is able to perform practical route operations as Tor, and initialize and observe attacks. This simulation platform gives us more confidence in investigating Tor related issues.

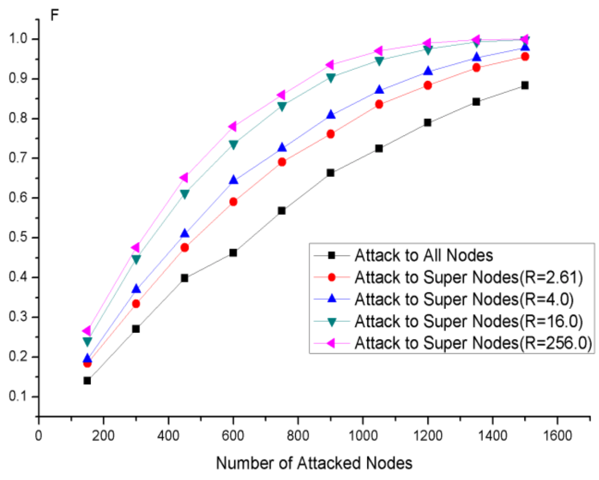


Figure 10.Results of Brute Force Attack.

There are a few limitations for our simulation platform. First, we do not perform the actually data encryption/decryption and transmissions over the network. The nodes only perform key features such as join, leave, path selection, etc. This simplifies the computation load, while not reflecting the real network complexity. For our investigation, network delivery does not affect our anonymity study. Second, the simulation system is driven by events such as the initialization/termination of connections. The important point is that these limitations do affect the investigation of super nodes in this paper. We can use the current simulation platform to examine brute-force attacks and malicious-node attacks. The simulated results reflect the same trend to the real Tor algorithms and setting. We are improving the simulation platform to support more features as the real Tor system for our future investigation.

6.2 Simulated Attacks

6.2.1 Evaluating Brute-Force Attacks

We have evaluated the effect of super nodes under brute-force attacks with 3000 relay nodes. In the first test, a brute-force attack aims at all nodes with the same probability, just as a normal brute-force attack without the knowledge of super nodes. In the second test, attackers aim at super nodes. The number of super nodes is set to 1500, half of the network size.

We define a *communication failure rate F* to evaluate the effect of brute-force attacks. When a node on path is attacked, we consider the path is disabled. We define *F* as the number of failed paths over the number of all paths. The test results are shown in Fig.10. The selection ratio of super nodes over normal nodes ***R*** is chosen to be 2.6, 4, 16, and 256 to show their effects on the failure rate. When the selection ratio ***R*** is 2.6 (the minimum seen in our dataset), the failure rate of attacking super nodes (the second curve from the bottom) is already much higher than that of attacking random nodes. In one extreme case, blocking all super nodes (half nodes of the system) makes the failure rate close to 100%. In another case, when attacking 150 super nodes (only 5% of all nodes in the system), the failure rate is already near 20%. Furthermore, when ***R*** becomes larger, the failure rate F rises rapidly (top three curves). For example, when attacking 750 super nodes and R=16, the failure rate is higher than80%. This will clearly damage the system performance and the user experience.

6.2.2 Malicious-Node Attack

We have presented our analysi sis of p_m, p_r and p_t in Fig.6. We also use the simulation platform to verify the theoretical analysis. The results are also shown in Fig.6. From the comparison, the simulated p_m, p_r and p_t curves fit the theoretical analysis very well, and confirm that the theoretical analysis to p_m, p_r and p_t is accurate.

7. CONCLUSIONS AND FUTURE WORK

In this paper, we have confirmed the existence of super nodes in Tor through experiments and analysis. We have then revisited the basic anonymity measure, analyzed several attacks exploiting super nodes, and developed a new attack.

For our follow-up work, we will further investigate the practical and theoretical impacts of these attacks exploiting the existence of super nodes. We will also investigate mitigation solutions to address attacks exploiting super nodes, e.g., developing distributed lightweight trust and supervising mechanisms. Although the super nodes discussed here are Tor relays, such issues are also applicable to many other p2panonymous communication systems with similar properties. We will look into more general cases and conduct more theoretical analysis.

8. ACKNOWLEDGMENTS

Prof. Yingfei Dong's current research is supported in part by US NSF Grants CNS-1041739, CNS-1120902,CNS-1018971, and CNS-1127875. Any opinions, findings, and conclusions or recommendations expressed in this paper are those of the authors and do not necessarily reflect the views of National Science Foundation.

This research is supported by the 973 National Basic Research Program of China (2007CB311102).

Funded by Tsinghua National Laboratory for Information Science and Technology (TNList) Cross-discipline Foundation.

9. REFERENCES

[1] Vasserman, E., Jansen, R., Tyra, J., Hopper, N., and Kim, Y. 2009. Membership-concealing overlay networks. In Proc. of the 16th ACM conference on Computer and Communications Security, New York, NY, USA.

[2] Nambiarand, A., and Wright, M. 2009. Salsa: A Structured Approach to Large-Scale Anonymity. In Proc. of the 13th ACM conference on Computer and Communications Security, New York, NY, USA.

[3] Dingledine, R., Mathewson, N., and Syverson, P. 2004.Tor: the second-generation onion router. In Proc. of the 13th conference on USENIX Security Symposium, Vol. 13, 2004.

[4] Tor Metrics, http://metrics.torproject.org/, 2011.

[5] BGP Table, http://bgp.potaroo.net/as2.0/bgptable.txt, Jan. 2011

[6] Tor directory protocol, version 3,http://tor.eff.org/svn/trunk/doc/spec/dir-spec.txt, 2010.

[7] Serjantov, A., and Danezis, G. 2002. Towards an information theoretic metric for anonymity. In Proc. of Privacy Enhancing Technologies workshop (PET'02), LNCS 2482, San Francisco, CA, USA, 2002:41-53.

[8] Díaz, C., Seys, S.,Claessens, J., and Preneel, B. 2002. Towards measuring anonymity. In Proc. of Privacy Enhancing Technologies workshop (PET'02), LNCS 2482, San Francisco, CA, USA, 2002:54-68.

[9] Reiter, M. K., and Rubin, A. D.1998. Crowds: Anonymity for web transactions, ACM Transactions on Information and System Security (TISSEC) 1(1), 66-92.

[10] Berthold, O., Federrath, H., and Köpsell, S.2001. Web MIXes: A system for anonymous and unobservable Internet access. Designing Privacy Enhancing Technologies: Design Issues in Anonymity and Observability. Springer-Verlag, LNCS2009, pp. 115-129.

[11] WikiLeaks and Julian Paul Assange. 2010. The New Yorker. http://www.newyorker.com/reporting/2010/06/07/100607fa_fact_khatchadourian?currentPage=all#ixzz0pWdlAepe.

[12] Hughes, D., and Shmatikov, V. 2004.Information hiding, Anonymity and Privacy: A modular approach. Journal of Computer security, 2004, 12(l):3-36.

[13] Shields, C., and Levine, B. N. 2000.A protocol for anonymous communication over the Internet. In Proc. of the 7th ACM conference on Computer and Communications Security, Athens, Greece, 2000:33-42.

[14] Chaum, D. 1981. Untraceable Electronic Mail, Return Addresses, and Digital Pseudonyms. Communications of the ACM, 1981, 4(2): 84-88.

[15] Wikileaks, http://en.wikipedia.org/wiki/WikiLeaks.

[16] Linode, http://www.linode.com.

[17] Li, F., Luo, B., Liu, P., and Chu, C. H. 2010. A Node-failure-resilient Anonymous Communication Protocol through Commutative Path Hopping. In Proc. of the IEEE INFOCOM, 2010.

[18] Fu, X. W., Zhu Y., Graham, B., Bettati, R., and Zhao W. 2005. On flow marking attacks in wireless anonymous communication networks. In Proc. of IEEE Inter. Conf. on Distributed Computing Systems (ICDCS), April 2005.

[19] Murdoch, S. J., and Danezis, G. 2006. Low-cost traffic analysis of tor. In Proc.of the IEEE Security and Privacy Symposium (S&P), May 2006.

[20] Ling, Z., Luo, J. Z., Yu, W., Fu, X. W., Xuan, D., and Jia, W. J. 2009. A New Cell Counter Based Attack Against Tor. In Proc. the 16th ACM conference on Computer and Communications Security, New York, NY, USA.

[21] Evans, N. S., Dingledine, R., and Grothoff, C. 2009. A practical congestion attack on Tor using long paths. In Proc. of USENIX Security Symposium, USENIX, Aug. 2009.

[22] D. M. Goldschlag, M. G. Reed, and P. F. Syverson. 1999. Onion routing. *Communications of the ACM*, 42(2):39–41, 1999.

Smart Metering De-Pseudonymization

Marek Jawurek
SAP Research
Vincenz-Priessnitz Str.1
Karlsruhe, Germany
marek.jawurek@sap.com

Martin Johns
SAP Research
Vincenz-Priessnitz Str.1
Karlsruhe, Germany
martin.johns@sap.com

Konrad Rieck
Technische Universität Berlin
Franklinstrasse 28/29
Berlin, Germany
konrad.rieck@tu-berlin.de

ABSTRACT

Consumption traces collected by Smart Meters are highly privacy sensitive data. For this reason, current best practice is to store and process such data in pseudonymized form, separating identity information from the consumption traces. However, even the consumption traces alone may provide many valuable clues to an attacker, if combined with limited external indicators. Based on this observation, we identify two attack vectors using anomaly detection and behavior pattern matching that allow effective de-pseudonymization. Using a practical evaluation with real-life consumption traces of 53 households, we verify the feasibility of our techniques and show that the attacks are robust against common countermeasures, such as resolution reduction or frequent re-pseudonymization.

1. INTRODUCTION

The deployment of Smart Metering—the digital recording and processing of electricity consumption—is ever increasing. A Smart Meter is an electrical meter that records a fine-grained consumption trace of a household and sends it to the respective electricity supplier. These consumption traces, in contrast to traditional single annual consumption values, allow the realization of time-of-use tariffs and demand response schemes.

This flexibility, however, comes at a price. Every activity that takes place in the household and makes use of electrical appliances is reflected in the consumption trace. In consequence, Smart Metering has repeatedly been called a privacy invasion into households [7, 8] and a large body of previous work [5, 6, 11, 12, 14, 15, 20] has been concerned with inferring private information from energy consumption traces.

Based on the identified privacy implications, there is consensus that consumption data of Smart Metering needs to be adequately protected. Such protection entails the protection during storage by the supplier and during the use of the data by the supplier and 3rd party contractors. Pseudonymization of consumption traces is considered an effective defense against privacy attacks, as it allows for unlinking the identity of the household and its consumption trace. The consumer's identity can be stored independently from consumption traces, only linked by the pseudonym. In such a scenario, the privacy-invading methods developed in previous work can only be applied by the owner of both, the identity database and the consumption traces.

An attacker faces two problems, if he has only access to pseudonymized traces: First, deduction from pseudonymous consumption traces is error-prone as no identity information can be used as contextual data. Second and more important, all information inferred from consumption traces can not be attributed to a specific household due to the unlinkability introduced by pseudonymization. This makes consumption traces and its contained information unattractive for targeted abuse and apparently the consumers' privacy is protected.

In this paper, we develop two attack vectors targeting the privacy of pseudonymized consumption traces. The first attack allows to create a link between a household's identity and its consumption trace, and therefore enables an attacker to undo pseudonymization. If successful, this attack allows all existing deduction attacks to be applied again. The second method enables an attacker to track the origin of a consumption trace across re-pseudonymization or across different databases. For conducting these attacks in practice, we provide a data analysis framework that allows an attacker to apply either method to consumption databases.

The paper's main contributions are as follows:

1. An abstract definition of attack vectors on the unlinkability of pseudonymous Smart Metering consumption traces.
2. A machine learning framework for the analysis of consumption traces and subsequent execution of aforementioned attack vectors.
3. Experimental findings about the anomaly detection in consumption traces and the tracking of consumption traces across pseudonyms.
4. An evaluation of different mitigation techniques with respect to their effectiveness against those attacks.

The rest of this paper is structured as follows: In Section 2 we provide an overview of the terminology used in this paper. Section 3 describes the two attack vectors that we identified for de-pseudonymization. In Section 4, we present a data analysis framework for conducting these attacks in practice. Section 5 shows our experimental results regarding the applicability of our attacks. Approaches for limiting linkability like reduced resolution, frequent re-pseudonymization and cryptography solutions are discussed in Section 6. Finally, in Section 7 we discuss related work before we conclude in Section 8.

2. TERMINOLOGY AND ASSUMPTIONS

This paper's understanding of identity, pseudonymity and other privacy related terms is based on terminology definitions provided by Pfitzmann and Hansen [19].

In the following we will use the term consumption trace for a recording of electric consumption in discrete time slots of equal length. The resolution of a consumption trace is the number of time slots per day. The term consumer stands for the household that makes use of the utility that is measured by a Smart Meter. The subject supplier stands exemplary for any subject (actual supplier or grid operator) that participates in Smart Metering and has legitimate interest in consumption traces, e.g. for billing electricity consumption. The Smart Meter records the consumption constantly and communicates it to the supplier. The Smart Meter, respectively the consumption trace, are attributed with a pseudonym for the consumer. This pseudonym is pre-arranged by the supplier and non-public.

The supplier stores consumption traces by pseudonym in a consumption trace database. The supplier accesses this database for billing or analysis in general. Contractors of the supplier also have access to this database but not to the identity database. Contractors provide analysis services to suppliers based on pseudonymized consumption traces. For the purpose of invoice creation the supplier owns another database, the identity database, that connects the consumer's identity with the pseudonym.

Optionally, a supplier may re-pseudonymize a consumption trace repeatedly, creating a 1 to n relationship between identities and pseudonyms. Consumers can switch suppliers which leads to the following situation: They have an old consumption trace, identity information and linking pseudonym in the old supplier's databases and also a current consumption trace and new pseudonym together with their identity information in their new supplier's databases. Contractors and suppliers may behave semi-honest, i.e, they stick to the protocols and respective laws but try to learn as much about consumers as possible. Our attacker model includes contractors, suppliers but also external malicious agents that might illegally obtain consumption traces.

3. ATTACK DESCRIPTION

The final goal of our attacker model is to create a link between the identity of consumers and their energy consumption. To this end, we present two different attacks that can be applied to achieve this goal. We name the attacks: *linking by behavior anomaly* and *linking by behavior pattern.*

Once a link between a consumption trace and a specific household has been established all information contained in the trace can be attributed to this household. If, on the other hand, a consumption trace cannot be linked to one household with a significant higher probability than to another household, this means that the data and its contained information cannot be attributed to a single household. In this case, the contained information would not have a privacy impact on its origin. Thus, linkability is a sufficient condition for privacy loss in Smart Metering.

3.1 Linking by Behaviour Anomaly

Linking by behavior anomaly (LA) can be used by the attacker to link either an identity to a consumption trace or two consumption traces with each other (see Figures 1 and 2). This is accomplished by *identifying and correlating anomalies* in both data sources that occur at the same time.

We characterize an anomaly as a series of unusual events, where an event is some consumer behavior that is reflected in the energy consumption of the respective household. The rarity of an anomaly is based on different factors: Length of the series, the resolution of the time stamp (day, hours or minutes) and rarity of the singular events among the population of consumption traces under consideration. Length and resolution make up for singular events that happen very often among the population: Leaving home or coming home at slightly different times during the course of one week. On the other hand, there are events that neither require a series nor a high resolution: moving in/out, death/birth or holidays. A series of events for a low resolution can be observed if household inhabitants leave every weekend or stay at home always at specific work days. Here the rarity originates from the length of the series.

With respect to linking a consumption trace to an identity, LA can be used in both ways. Either identifying the household for a consumption trace or vice versa. It really depends on the final purpose of the identification which approach is taken, whether specific households should be targeted (like both examples in Section 3.3) or specific consumption profiles are of interest. The main requirement for this attack is to have two data sources that overlap for the time interval where an anomaly has been identified.

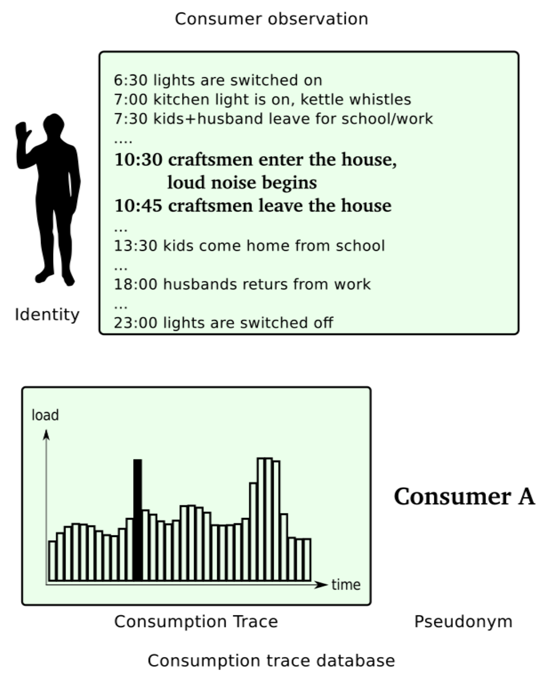

6:30 lights are switched on
7:00 kitchen light is on, kettle whistles
7:30 kids+husband leave for school/work
....
10:30 craftsmen enter the house, loud noise begins
10:45 craftsmen leave the house
...
13:30 kids come home from school
...
18:00 husbands returs from work
...
23:00 lights are switched off

Figure 1: Behavior and consumption anomaly

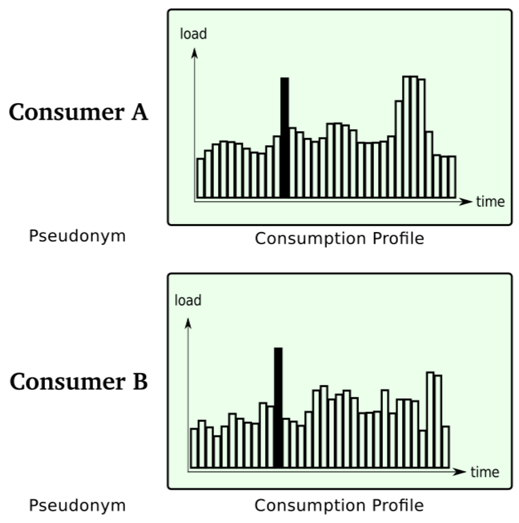


Figure 2: Anomalies in two consumption traces

3.2 Linking by Behavior Pattern

The goal of linking by behavior pattern (LP) is to link different pseudonyms of one consumer (see Figure 3). There are two reasons for one consumer to have different pseudonyms attached to his identity: Either one supplier re-pseudonymizes his consumption trace database or the consumer has consumption traces in different databases with different pseudonyms respectively. The latter happens when a consumer changes his supplier. The old supplier still possess the consumption trace with one pseudonym and the new supplier starts to collect a consumption trace under a new pseudonym. This is equivalent to tracking consumers across different databases.

The former case, re-pseudonymization, means that a consumer's traces are stored under the pseudonym A in time interval t and stored under pseudonym B in time interval $t + 1$ in the same database. This method could be applied by suppliers to prevent the de-pseudonymization of many years of consumption profiles under the same pseudonym in case one pseudonym is de-pseudonymized.

For this attack the attacker requires a database of pseudonymized consumption traces containing the pseudonym A. He tries to find a consumption trace with pseudonym B that has been created by the same consumer. Then the attacker can link the pseudonym A with pseudonym B.

In contrast to the LA attack, this attack can be applied even if the data sources do not overlap in time. This is because fundamental patterns in consumption are identified and subsequently looked for in the other data source. This means, that we can either consider consumption slices from two different consumption trace databases or two consumption traces from the same database but from different time intervals.

The feasibility of such an attack would also imply that everyone possessing current consumption traces and links to consumer identities can harvest all consumption traces that have been published (even in anonymized form) in the past or can be obtained for the past. On the other hand, legitimate holders of old consumption trace databases and

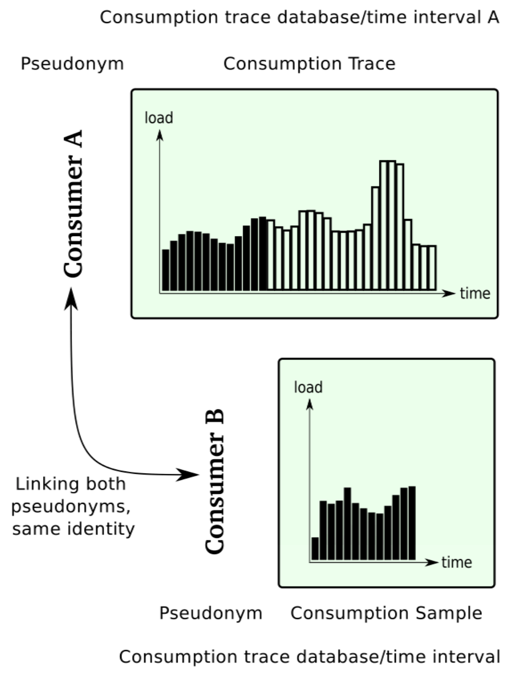

Consumption trace database/time interval A
Pseudonym
Consumption Trace
Consumer A
load
time
Linking both pseudonyms, same identity
Consumer B
load
time
Pseudonym
Consumption Sample
Consumption trace database/time interval B

Figure 3: Identifying consumption traces with the same origin

the corresponding identities are able to "de-anonymize" data traces that are publicized in the future.

3.3 Exemplified Attacks

There is a multitude of attack scenarios that involve linking Smart Meter data and contained information to their origin household. We herein provide two examples:

Fake sick-day disclosure.

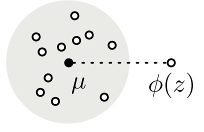
An employer could attempt to spy on his employees. In particular, he would like to know whether employees really spend their sick days at home. In order to do that, the employer obtains pseudonymized consumption traces for consumers in the geographic region of his employees' households. Using a correlation of vacation/company travel data with the consumption trace he links his employees' identities to the respective pseudonymized consumption traces. Then the employer can try to derive employee behavior on sick days from the employee's consumption trace.

Absence pattern deduction.

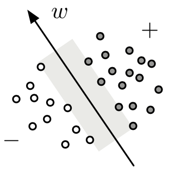
Another motivation for linking a household to its consumption trace is preparing a burglary [17]. Once a burglar has identified a worthwhile victim in the physical world, the burglar would like execute his plans undisturbed by the inhabitants. The burglar therefore performs an observation of the household in question and their weekend behavior. He simply finds out whether they stay at home or leave for the weekend over the course of several weeks. By correlating this information with a consumption trace database this household's trace can be identified in the database. Now, repeating long-term absence patterns of this household can be found, e.g. for a regular family meeting or a time share, and subsequently the burglary can be scheduled for such a date. Traditionally, a burglar would need to observe a household for years to get these information. The linking of household identity and its consumption trace, however, allows him to tap into a wealth of information about a long time-frame of the household in question.

4. DATA ANALYSIS FRAMEWORK

So far we have studied the linking of consumption traces and consumer identities in an abstract manner. For conducting the presented attacks in practice, we now introduce a data analysis framework. This framework builds on concepts of machine learning and allows us to analyze consumption traces geometrically. To this end, the consumption trace of a consumer is mapped to a high-dimensional feature space, such that it can be analyzed by standard techniques of machine learning. In this geometric representation, the *linking by behavior anomaly* can be achieved using geometric anomaly detection, where unusual events are identified by a large distance from normal activity. Similarly, the *linking by behavior pattern* can be carried out using geometric classification, where the behavior of one consumer is separated from all other users in the feature space. Figure 4 illustrates this geometric interpretation of the two attacks.



(a) Anomaly detection



(b) Classification

Figure 4: Schematic depiction of data analysis: (a) distance to mean; (b) separating hyperplane.

Before presenting this data analysis framework, we introduce some basic notation. We refer to the known consumption trace of a consumer by $X = \{x_1, \dots, x_n\}$, where X covers n days and each x_i corresponds to the measurements of one day. Depending on the resolution d of Smart Metering, each x_i is represented by a vector of d dimensions

$$x_i = (m_1, \dots, m_d),$$

where m_j is the consumption measured at the j-th time slot of the Smart Metering resolution. Moreover, if we consider a grid of g bins associated with consumption values, we say that m_j falls into bin k, if m_j has the smallest difference to the consumption value associated with the k-th bin.

4.1 A Binary Feature Space

Mapping consumption traces to a vector space may seem trivial at a first glance, as the measurements of a day are already represented as a d-dimensional vector. However, for discriminating different patterns in this data, we require a more advanced representation that emphasizes the characteristics of each consumer and provides an expressive basis for application of machine learning.

Depending on the setup of electronic devices in a household, the consumption of a consumer changes between different states. Devices are switched on and off; thereby the consumption moves from one state to another. This discrete behavior is illustrated in Figure 5, which shows the consumption of one user over the period of one week on a fixed grid. Dark entries in the grid indicate frequent occurrences of a consumption value. It is notable that the consumption is neither a continuous nor a smooth function and several discrete states can be observed. For example, between 10:00 o'clock and 11:00 o'clock the consumption matches one of three possible states at roughly 300, 900 and 1500 W/h respectively.

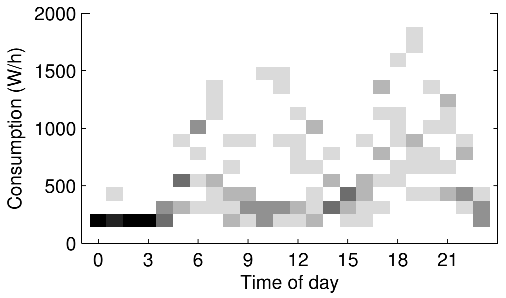


Figure 5: Grid representation of a consumption trace. Dark color indicates frequent occurrences.

Inspired by this observation, we construct a feature space that specifically reflects these states of consumption data. In particular, we employ a grid of g bins that spans the range of observable consumption values. Using this grid we define an indicator function $I_{j,k}$

$$I_{j,k}(x) = \begin{cases} 1 & \text{if } m_j \text{ falls into bin } k \\ 0 & \text{otherwise.} \end{cases} \tag{1}$$

that returns 1, if the j-th measurement of a day x falls into the k-th bin of our grid, and 0 otherwise.

If we compute the indicator function $I_{j,k}$ for all d measurements of a day and all g bins of the grid, we obtain a mapping

$$\phi : X \mapsto \mathbb{R}^{d \cdot g}, \quad \phi(x) = \big(I_{j,k}(x)\big)_{\substack{1 \le j \le d \\ 1 \le k \le g}} \tag{2}$$

that maps the consumption of one day to a binary feature space with $d \cdot g$ dimensions. For example, if we have one measurement per hour, that is $d = 24$, and use a grid with $g = 100$, we embed the data in a feature space of 2,400 binary features, each corresponding to a particular consumption value at a particular hour. In contrast to a naive representation with d dimensions, this feature space describes states independently of absolute differences. That is, each state change induced by the consumer is treated equally, whether it involves switching on a small desk lamp or a powerful washing machine. For the experiments in Section 5, we consider $d = 24$ and make use of a grid with 100 bins of consumption values.

4.2 Anomaly Detection

By embedding the consumption traces into an expressive feature space, we are able to phrase our attacks in terms of geometric relationships between data points. For determining unusual activity in the data of a consumer we employ a standard technique for detecting geometric outliers. Given a consumption trace of n consecutive days, we first learn a *profile* of normal activity by computing the mean μ of the data in the feature space as follows

$$\mu = \frac{1}{n} \sum_{i=1}^{n} \phi(x_i). \tag{3}$$

The profile μ captures the states shared by the majority of the consumption traces. As each vectors $\phi(x_i)$ contains only binary values, each dimensions of μ can be interpreted as the probability for observing a particular grid value at particular time of the day. Geometrically, the deviation of a day z from this profile can be determined by simply computing the distance

$$d(z) = ||\phi(z) - \mu||. \tag{4}$$

Note that $d(z)$ corresponds to the Euclidean distance in the vector space and can be efficiently computed with standard software libraries. This generic approach to anomaly detection is illustrated in Figure 4(a). If we notice a large distance $d(z)$ for a day z, some of the consumption states of this day differ from normal activity and z is likely to contain an anomalous event.

This technique for computing profiles can also be applied to compare different sets of consumption traces. For example, if we have two reference sets X_1 and X_2 from the same consumer, we can compute two mean values μ_1 and μ_2 and compare the distance to both. This setting allows us to study different classes of days during analysis, as shown in Section 5 for weekdays and weekends.

4.3 Classification

For *linking by behavior pattern*, we aim at inferring patterns from the consumption trace of a consumer. However, we are not interested in modelling the complete behavior of a consumer, but determining patterns that discriminate his behavior from others. Thus, we employ the technique of classification and learn a discrimination between users. A robust method for learning such a discrimination is a Support Vector Machine (SVM) [2, 16]. An SVM basically determines a hyperplane in the feature space that separates two classes with maximum margin. This geometric concept is illustrated in Figure 4(b). The hyperplane is constructed as a linear combination of the training data and separates the consumption trace of one consumer c from all others. Formally, this hyperplane is given by a direction vector

$$w_c = \sum_{i=1}^{n} y_i \alpha_i \phi(x_i) \tag{5}$$

and an offset term b_c, where $y_i \in \{-1, +1\}$ are training labels indicating whether day x_i corresponds to consumer c and α_i are the learned coefficients.

To account for multiple consumers, we make use of the *one-against-all approach* and learn a hyperplane for each consumer separating him from the rest of users. The discrimination function for each consumer is then given by

$$h_c(z) = \langle \phi(z), w_c \rangle + b_c. \tag{6}$$

The function $h_c(z)$ reflects the distance from day z to the hyperplane of consumer c. The more consumption states and patterns are shared with c, the higher $h_c(z)$ gets. Hence, if we want to link a day z to a consumer, we simply assign it to those consumer c with the largest value for $h_c(z)$.

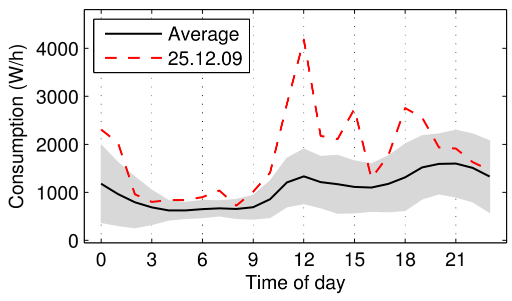
Similarly to the anomaly detection, there exists several efficient libraries for computing SVMs. In our experiments, we make use of LibLinear [4]—a library capable of learning with millions of dimensions and data points.

5. EXPERIMENTAL EVALUATION

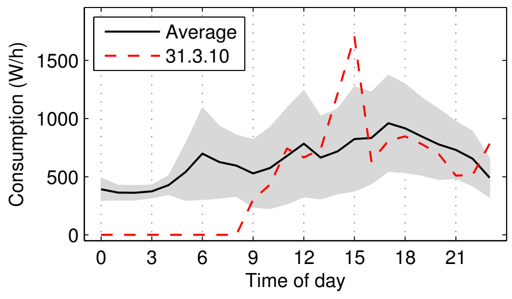
To study the impact of our attacks in practice, we conduct experiments using anonymized consumption traces for 53 households. The data stretches over 221 days and has a resolution of one value per hour. The goal of the experiments is to demonstrate the efficacy of the two attacks described in Section 3. However, due to privacy reasons we do not have identity data for the consumption traces and therefore cannot fully implement the LA attack. Yet, we can identify significant anomalies in the consumption trace that could greatly help in linking identities to pseudonyms.

5.1 Identification of Anomalies

In our first experiment, we apply the technique of anomaly detection to each of the 53 consumers. We are interested in identifying days that stand out of regular energy consumption and might provide a good basis for linking the consumption data with an external data source.



(a) Anomaly detected for consumer 21.



(b) Anomaly detected for consumer 49.

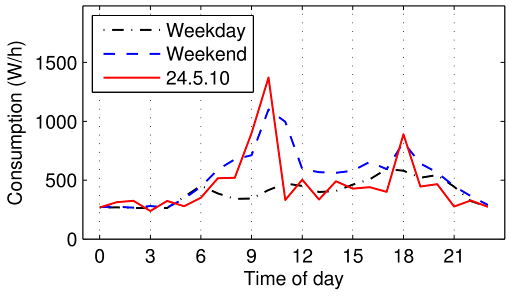
Figure 6: Exemplary anomalies for two consumers. The shaded area indicates the standard deviation.

Figure 6(a) shows exceptionally high energy consumption throughout the day, probably in the course of Christmas preparations. Depending on the amount of time this consumption trace spans this could mean different things: If the trace spans several years (which our specific data source does not) this would indicate that this consumer has not has had such extensive Christmas preparations because he/she previously went away for Christmas or celebrates Christmas for the first time at home. As this particular trace only spans approximately 7 months, this deviation just indicates that the 25th of December means something special to this household, which could in turn indicate that it is Christian. Depending on the context of this household this could mean incriminating information.

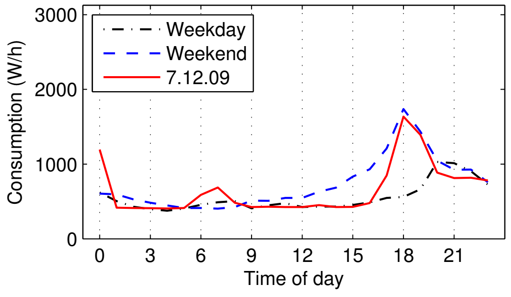
Figure 6(b) displays how a consumer apparently starts energy consumption shortly before 9 o'clock. This may indicate that the consumer has moved in and started the first electric devices in his new apartment. In the afternoon exceptionally high load can be observed, consistent with the use of machines by craftsmen. If one could correlate this data with data sources that hold information about moving households in this region this could lead to the identification of the inhabitants.

While we have shown only two strong anomalies from our data set, several others can be identified for the consumers. Provided external reference data, it is trivial to correlate these anomalies with unusual events and there is a realistic chance of unlinking pseudonyms.

As mentioned in Section 3.1, anomalies can also be identified using different profiles of consumption. In this second experiment, the consumption on workdays and weekends are analyzed to determine whether the household inhabitants stay or leave home. In particular, we compare the profile of weekends and workdays to identify workdays that match the consumption behavior of weekends.



(a) Profiles for consumer 12 and "day-off".



(b) Profiles for consumer 40 and "day-off".

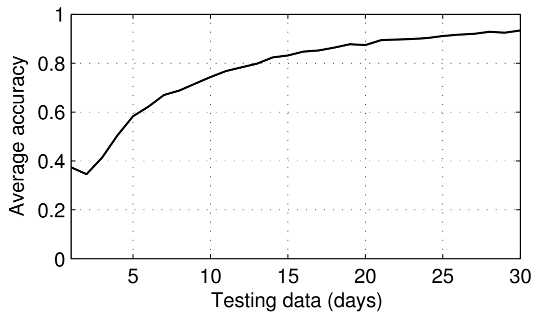
Figure 7: Exemplary weekday and weekend profiles. The red line indicates a potential "day off".

Figure 7(a) displays how usual workday and weekend profile look like for a consumer. For the 24th of May however, we can identify a day that matches a weekend day judging by its energy consumption but is a Monday (workday). Figure 7(b) shows the same for another consumer for the 7th of December 2009 which is a Thursday.

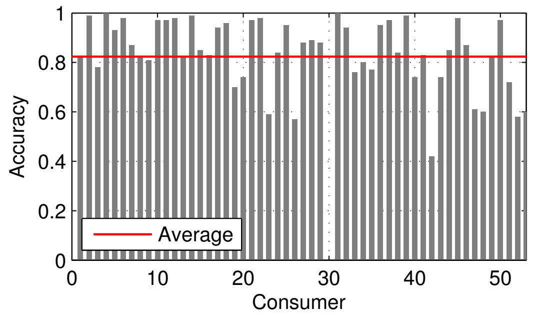
It is notable that for the given resolution of Smart Metering accurate profiles can be learned for the consumption data. While we have shown examples for comparing weekends and weekdays, several other scenarios exist that would allow to further structure and analyze the electricity consumption of a household. Together with the anomaly detection, these analysis steps clearly allow to narrow down the search for a particular identity and help to unlink its pseudonym.

5.2 Linking by Behavior Pattern

In this experiment we conduct the LP attack by using the classification technique from Section 4.3 to determine whether consumption traces have the same origin household. We use one time interval of our consumption trace database for training our machine learning framework and subsequently provided it with test data from a different, non-overlapping time interval. The algorithm implemented by the framework then tries to link consumption traces that behave similarly. We measure its accuracy by the relative frequency of correct linking decisions.



(a) Accuracy for varying size of test data.



(b) Per-consumer accuracy for 14 days of test data.

Figure 8: Classification accuracy for varying sizes of test data (a) and individual users (b).

Figure 8(a) displays linking accuracies for training data of 60 days in dependency of test data size, where we assume fixed pseudonyms during the given time spans. The graph steadily climbs to over 90% accuracy for 30 days of test data. Figure 8(b) represents a breakdown of the linking accuracy for testing data of 14 days. For several pseudonyms an almost perfect unlinking is possible and on average an accuracy of 83% is attained, corresponding to 5 correct identifications out of 6 consumers. Note that pseudonym 30 undergoes significant perturbations over the course of our data which leads to repeated mis-classification and subsequently zero accuracy.

Figure 9 displays the accuracy of our approach depending on the sizes of the training set and the test data in days. One can see, that the size of the test data has a slightly stronger impact on the accuracy then the size of the training data. Overall, the accuracy reaches approximately 83% if the training and test data is larger than 28 days. As a result, our attack is even effective if a re-pseudonymization is conducted every month.

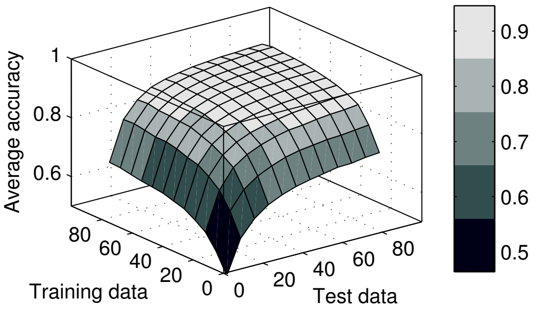


Figure 9: Impact of training and testing data size on classification accuracy.

6. MITIGATION TECHNIQUES

We finally investigate in this section three mitigation alternatives and their ability to contain and mitigate the aforementioned attacks.

6.1 Lower Resolution

A simple yet promising approach to mitigating our attacks is to lower the resolution of Smart Metering. The idea behind is that consumption traces are blurred and therefore anomalies and patterns are covered. In order to evaluate this mitigation technique we scale the consumption traces down in several steps and executed our experiments on the down-scaled data.

Figure 10 represents our anomaly detection for resolutions of 6, 3 or 1 value per day for the anomaly that was identified in Figure 6(a). One can see that the anomaly remains visible, even if the Smart Meter only records one value per day. The reason for this is that the anomaly spanned a larger part of the day and hence had a high impact on the total energy consumption of that day. Figure 11 shows an anomaly that behaves differently. While for resolutions of 6 or 3 values per day the anomaly still can be identified, it is not recognizable in the 1 value per day.

Regarding the LP attack the reduction of Smart Metering resolution has a bigger effect. Figure 12 shows the linking accuracy in dependency of the test data size for different resolutions. In contrast to Figure 9 the linking accuracy drops significantly with a reduction of the resolution. While the accuracy still reaches almost 70% for 8 measurements per day it drops to approx. 4% for one measurement a day.

These results show that a reduction of the Smart Metering resolution has mixed effects on our attacks. For anomaly detection (and subsequent anomaly linking) the attacker will probably be still quite successful if he manages to find

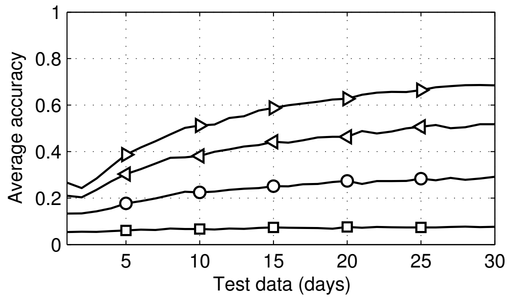


Figure 12: Classification accuracy with different resolution. The accuracy is given for resolutions of 8 (▷), 6 (◁), 3 (○) and 1 (□) value per day.

anomalies that have a big impact on the total energy consumption. However, short bursts of energy consumption will not be recognizable at low resolutions. Moreover, our results show that the LP attack can be successfully mitigated. The linking accuracy drops significantly with every reduction of the resolution.

Whether lower resolutions are a viable mitigation also depend on the supplier's requirements: The resolution of real-time tariffs and time-of-use prices is limited by the chosen Smart Metering resolutions.

6.2 Frequent Re-pseudonymization

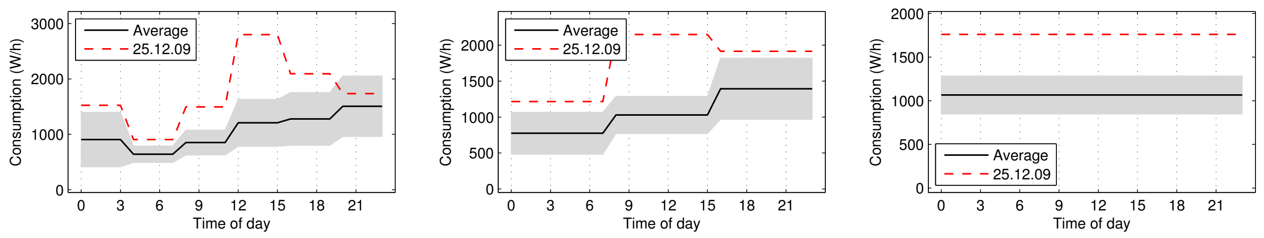
Re-pseudonymization could be considered another mitigation technique. The holder of the identity and consumption databases introduces new pseudonyms for identities every now and then. Subsequently arriving consumption trace items of a household will be recorded under the new pseudonym. This leads to the effect that a holder of only the consumption trace database has only short intervals of data per pseudonym. Which means, that the training set for the LP attack is limited by the re-pseudonymization time frame.

If we assume that the attacker tries to track origins of consumption traces across re-pseudonymization he would try to match two data sets of different pseudonyms to determine whether they belong to the same origin. If we assume that the re-pseudonymization time frame is constant then the size of the training set and test set are the same. The potential effect on the LP attack can be seen in Figure 9.

Even if the re-pseudonymization time frame has only 20 days we can link two pseudonyms of the same identity with 80% accuracy. Thus, for the method to be effective one would need to re-pseudonymize in very short intervals. There are two major drawbacks of re-pseudonymization: First, analysis over frequently re-pseudonymized consumption traces can only span intervals of the re-pseudonymization time frame. Analysis by contractors that requires long-term consumption data is not possible because they would need information about the pseudonymization for that. Second, frequent re-pseudonymization incurs an overhead of storing the linking between the different pseudonyms and the identity.

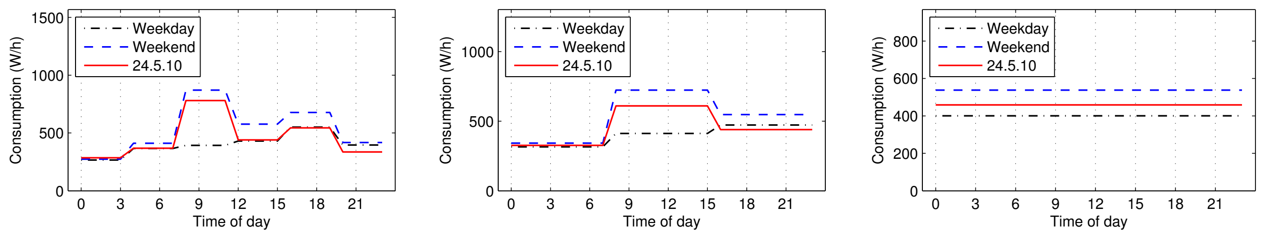
6.3 Privacy-preserving Techniques

Another mitigation technique is the prevention of transmitting and storing consumption traces in the first place. The approaches described in [9] or [21] reduce the amount



(a) 6 values per day. (b) 3 values per day. (c) 1 value per day.

Figure 10: Anomaly detection with different resolution. The detected anomaly from Figure 10(a) is shown.



(a) 6 values per day. (b) 3 values per day. (c) 1 value per day.

Figure 11: Weekday and weekend profiles with different resolution. The profiles from Figure 7(a)) are shown.

of data that is effectively intelligible by the suppler to one value per reporting interval. This value represents the price for the energy consumption of this day. It allows application of high-resolution time-of-use or real-time prices and can be used to just report one value per year if desired. This way it combines the advantages of a very low resolution (one value per year) and the application of demand response tariffs.

It is noteworthy, that the single value per reporting interval does contain a weighting of the energy consumption by the applicable tariff. Therefore it contains more information than a single consumption value of similar low-resolution strategy. However, its applicability is constrained to billing and cannot be used for forecasting since the tariff skews the actual consumption.

7. RELATED WORK

Attacks on Smart Metering privacy.

The works [5, 6, 11, 12, 14, 15, 20] investigate in the broad area of consumption trace analysis and behavior analysis form energy consumption. The authors of [14], for instance, investigate the effectiveness of their developed behavior deduction using a NALM approach. Over the course of two weeks they conducted an experiment that collected electrical data and video surveillance of the inhabitants. Their developed system inferred behavior events and load events from the electrical data and evaluated the performance of behavior analysis with control data extracted from the video surveillance. Finally, they construct a sample disclosure metric that categorizes their behavior deductions into categories like presence, sleep schedule and others and rates the disclosure in those categories according to the ability of their behavior-extraction system. [11] on the other hand focuses on detecting and characterizing different appliances according to load signatures.

Mitigation by data prevention.

The following approaches are good at ensuring consumer privacy because no household specific consumption traces are stored at the supplier's and therefore the LA/LP attacks are not feasible:

In [1] a model for measuring privacy in Smart Metering is developed and subsequently two different solutions to ensure privacy are presented: A Trusted Third Party-based approach, where individual consumption profiles are aggregated at the third party and only sums are communicated to the supplier. The other approach attempts to mask consumption profiles by adding randomness to the actual profile with an expectation of the random distribution of zero.

In [18] another twofold approach is presented: The first solution employs a sophisticated Trusted Platform Module (TPM) in the Smart Meter to obtain signed tariff data from the supplier and calculate a trustworthy bill. The second solution makes use of the electrical grid infrastructure as a third party to anonymize up-to-date consumption values sent out constantly by Smart Meters.

In [13] the authors first perform an informal threat analysis of Smart Metering and provide a sketch for an attested Smart Meter architecture. Using virtualization, mandatory network access control and trusted computing techniques this architecture enables multiple applications to use the Smart Meter hardware and to work in a privacy-preserving and integer manner. The article identifies applications for billing the consumer very closely to the data origin (in the household) and applications that provide the consumer with a consumer portal. They achieve privacy-preserving Smart Metering billing by remote attestation of the billing software in the TPM of the Smart Meter.

A cryptographic approach to Smart Meter privacy has been presented in [9]. A privacy component homomorphically calculates the price locally in the household and only reports the final price and cryptographic proofs to the supplier. With the help of those proofs the correct calculation with the correct tariff can be verified. In contrast to [21] this work focuses on how it can be built into existing Smart Meter reporting protocols.

Another cryptographic approach very similar to [9] is described in [21]. It focuses on realizing a variety of different tariff types with a cryptographic solution. Both approaches have been jointly evaluated in Section 6.3 with respect to their effectiveness to the developed attacks.

Mitigation by anonymization.

In [3] the authors propose a system to separate the data flow from the Smart Meter into two flows: One high- and one low-frequency data flow. The low frequency flow is attributed with the household's identity and can therefore be used for billing. The high-frequency data flow, that tells more about the habits of household inhabitants, is transmitted anonymously by the Smart Meter. An escrow service, potentially provided by the manufacturer of the Smart Meter, authenticates the anonymous high frequency flow towards the utility so that trust can be placed in its authenticity. The escrow service can disclose the identity of the high-frequency data flow in case of abuse.

This approach anonymizes the high-frequency data flow but cannot mitigate our attack vectors. Our assumptions are that attackers have access to anonymous (from their point of view) consumption traces but still manage to create linkability using correlation with secondary data sources.

Mitigation by hiding.

In [10] the authors propose to use a 'Load Signature Moderator' (LSM) and batteries to mask consumption events that represent 'privacy threats'. The LSM either detects or is notified by the appliances of approaching consumption events and could apply different algorithms like hiding, smoothing or obfuscation to hide those events from the Smart Meter. The batteries serve as energy buffer and enable the smoothing of actual loads. Then the authors measure the achieved privacy protection using three metrics: relative entropy cluster classification based similarity and regression analysis with different battery capacities and their best-effort moderation algorithm.

Our attack vectors would indeed be mitigated by the proposed approach. However, as the authors mention the proposed approach could conflict with cost-saving consumption strategies. On another note, it is questionable whether consumer see that their privacy value offsets the costs of such a system. If combined with cost-saving strategies, this might however be a desirable solution for consumers.

8. CONCLUSIONS

We have presented two attack vectors on the unlinkability of pseudonymized Smart Metering consumption traces. The first attack *linking by behavior anomaly* attempts to link a household identity to a pseudonymous consumption trace by anomaly correlation. The second attack *linking by behavior pattern* attempts to trace the origin of a consumption trace across different pseudonyms (due to re-pseudonymization or due to storage in different databases) by using patterns in their electricity consumption. To demonstrate the impact of the two attack vectors, we have presented a data analysis framework which allows us to conduct the attacks in practice and which we apply to perform experiments on real consumption traces.

Our experiments indicate that the task of finding relevant anomalies in consumption traces for the *linking by behavior anomaly* is feasible and allows deduction of household behavior. Regarding the *linking by behavior pattern* attack our experiments suggest that tracking the consumption trace across different pseudonyms is also feasible and can be executed quite accurately. Finally, we analyzed different mitigation techniques like lower resolution, frequent re-pseudonymization or data prevention with respect to their ability to thwart our attacks: A lower Smart Metering resolution has a mixed/good effect on the LA/LP attacks respectively, frequent re-pseudonymization requires very frequent (more often than 20 days) changes of pseudonyms to have noticeable (less than 80% accuracy) effects on linkability. Data prevention, by privacy-preserving cryptographic approaches that calculate the price in the household, has the best effect on both attacks and is also the most flexible with respect to high resolution time-of-use and real-time tariffs.

We have shown that alleged unlinkability introduced by pseudonymity of consumption traces is not sufficient for consumer privacy. Using the right secondary data sources attackers can link pseudonymized consumption traces back to consumers or track consumers across different databases of consumption traces. To prevent a failure of Smart Metering due to consumer distrust solutions must be found, that allow legitimate calculations on consumption traces without endangering consumer privacy.

9. FUTURE WORK

As we lack control data for our *linking by behavior anomaly* attack we could not fully evaluate its practical impact. Future work in this area is thus the investigation of linkability with adequate secondary data sources for willing consumers.

Another interesting question is whether persons can be tracked across different residencies. New apartments/houses induce some change in consumption patterns but to some extend personal habits and preferences might still be encoded in the consumption trace. A research question could be whether there is a component in the energy consumption pattern that remains the same, i.e. that identifies the inhabitants even across residencies?

10. ACKNOWLEDGMENTS

Martin Johns' work in this paper was partly funded by the German Federal Ministry of Economics and Technology (BMWi) as part of the e-IKT project with reference number 01ME09012.

Marek Jawurek's work in this paper was partly funded by the German Federal Ministry of Economics and Technology (BMWi) as part of the MEREGIOmobil project with reference number 01ME09007.

References

[1] J.-M. Bohli, O. Ugus, and C. Sorge. A privacy model for smart metering. In *Proceedings of the First IEEE International Workshop on Smart Grid Communications (in conjunction with IEEE ICC 2010)*, 2010.

[2] N. Cristianini and J. Shawe-Taylor. *An Introduction to*

Support Vector Machines. Cambridge University Press, Cambridge, UK, 2000.

[3] C. Efthymiou and G. Kalogridis. Smart grid privacy via anonymization of smart metering data. *2010 First IEEE International Conference on Smart Grid Communications*, pages 238–243, 2010.

[4] R.-E. Fan, K.-W. Chang, C.-J. Hsieh, X.-R. Wang, and C.-J. Lin. LIBLINEAR: A library for large linear classification. *Journal of Machine Learning Research*, 9: 1871–1874, 2008.

[5] G. Hart. Nonintrusive appliance load monitoring. *Proceedings of the IEEE*, 80(12):1870 –1891, dec 1992. ISSN 0018-9219.

[6] G. W. Hart. Residential energy monitoring and computerized surveillance via utility power flows. *IEEE Technology and Society Magazine*, June 1989.

[7] W. Heck. Smart energy meter will not be compulsory. NRC Handelsblad (online), April 2009. `http://www.nrc.nl/international/article2207260.ece/Smart_energy_meter_will_not_be_compulsory`.

[8] A. Jamieson. Smart meters could be 'spy in the home'. Telegraph (UK) (online), October 2009. `http://www.telegraph.co.uk/finance/newsbysector/energy/6292809/Smart-meters-could-be-spy-in-the-home.html`.

[9] M. Jawurek, M. Johns, and F. Kerschbaum. Plug-in privacy for smart metering billing. *CoRR*, abs/1012.2248, 2010.

[10] G. Kalogridis, C. Efthymiou, S. Z. Denic, T. A. Lewis, and R. Cepeda. Privacy for smart meters: Towards undetectable appliance load signatures. In *2010 First IEEE International Conference on Smart Grid Communications*, pages 232–237. IEEE, 2010.

[11] H. Lam, G. Fung, and W. Lee. A novel method to construct taxonomy electrical appliances based on load signaturesof. *Consumer Electronics, IEEE Transactions on*, 53(2):653 –660, may 2007. ISSN 0098-3063.

[12] C. Laughman, K. Lee, R. Cox, S. Shaw, S. Leeb, L. Norford, and P. Armstrong. Power signature analysis. *Power and Energy Magazine, IEEE*, 1(2):56 – 63, mar-apr 2003. ISSN 1540-7977. doi: 10.1109/MPAE.2003.1192027.

[13] M. Lemay, G. Gross, C. A. Gunter, and S. Garg. Unified architecture for large-scale attested metering. In *in Hawaii International Conference on System Sciences. Big Island.* ACM, 2007.

[14] M. A. Lisovich, D. K. Mulligan, and S. B. Wicker. Inferring personal information from demand-response systems. *IEEE Security & Privacy*, 8(1):11–20, January 2010.

[15] A. Molina-Markham, P. Shenoy, K. Fu, E. Cecchet, and D. Irwin. Private memoirs of a smart meter. In *Proceedings of the 2nd ACM Workshop on Embedded Sensing Systems for Energy-Efficiency in Building*, BuildSys '10, pages 61–66, New York, NY, USA, 2010. ACM. ISBN 978-1-4503-0458-0.

[16] K.-R. Müller, S. Mika, G. Rätsch, K. Tsuda, and B. Schölkopf. An introduction to kernel-based learning algorithms. *IEEE Neural Networks*, 12(2):181–201, May 2001.

[17] D. Mulligan and J. Lerner. Taking the long view on the fourth amendment: Stored records and the sanctity of the home. Talk (online), January 2007. URL `http://www.truststc.org/pubs/318.html`.

[18] R. Petrlic. A privacy-preserving concept for smart grids. In *Sicherheit in vernetzten Systemen: 18. DFN Workshop*, pages B1–B14. Books on Demand GmbH, 2010.

[19] A. Pfitzmann and M. Hansen. A terminology for talking about privacy by data minimization: Anonymity, unlinkability, undetectability, unobservability, pseudonymity, and identity management. online, Aug. 2010. URL `http://dud.inf.tu-dresden.de/literatur/Anon_Terminology_v0.34.pdf`.

[20] A. Prudenzi. A neuron nets based procedure for identifying domestic appliances pattern-of-use from energy recordings at meter panel. In *Power Engineering Society Winter Meeting, 2002. IEEE*, volume 2, pages 941 – 946 vol.2, 2002.

[21] A. Rial and G. Danezis. Privacy-preserving smart metering. Technical report, Microsoft Research, November 2010. URL `http://research.microsoft.com/apps/pubs/?id=141726`.

SEMAGE: A New Image-based Two-Factor CAPTCHA

Shardul Vikram
Texas A&M University
College Station, Texas
shardul.vikram@tamu.edu

Yinan Fan
Texas A&M University
College Station, Texas
yinanf@neo.tamu.edu

Guofei Gu
Texas A&M University
College Station, Texas
guofei@cse.tamu.edu

ABSTRACT

We present SEMAGE (**SE**mantically **MA**tching ima**GE**s), a new image-based CAPTCHA that capitalizes on the human ability to define and comprehend image content and to establish *semantic relationships* between them. A SEMAGE challenge asks a user to select *semantically related* images from a given image set. SEMAGE has a two-factor design where in order to pass a challenge the user is required to figure out the content of each image and then understand and identify semantic relationship between a subset of them. Most of the current state-of-the-art image-based systems like Assira [20] only require the user to solve the first level, i.e., image recognition. Utilizing the semantic correlation between images to create more secure and user-friendly challenges makes SEMAGE novel. SEMAGE does not suffer from limitations of traditional image-based approaches such as lacking customization and adaptability. SEMAGE unlike the current text-based systems is also very user-friendly with a high fun factor. These features make it very attractive to web service providers. In addition, SEMAGE is language independent and highly flexible for customizations (both in terms of security and usability levels). SEMAGE is also mobile devices friendly as it does not require the user to type anything. We conduct a first-of-its-kind large-scale user study involving 174 users to gauge and compare accuracy and usability of SEMAGE with existing state-of-the-art CAPTCHA systems like reCAPTCHA (text-based) [6] and Asirra (image-based) [20]. The user study further reinstates our points and shows that users achieve high accuracy using our system and consider our system to be fun and easy.

Categories and Subject Descriptors

K.6.5 [**[Computing Milieux**]: Management of Computing and Information Systems - Security and Protection

General Terms

Security

Keywords

CAPTCHA, Semantic-based Interactional Proofs,Two-factor CAPTCHA

1. INTRODUCTION

New web applications and services emerge everyday in all areas of life. More people are getting used to having online services, such as email services, forums, and specialized interest groups. For the service providers, one important aspect to consider is to make sure that the services and resources are allocated to the targeted customers. Malicious usage of services, such as using a 'bot' to register legal accounts [9], can take up valuable resources and distribute malicious information thereafter. Thus it is important for the service provider to be able to distinguish a bot from human users, and CAPTCHA systems are widely used for this purpose.

CAPTCHA stands for "Completely Automated Public Tests to tell Computers and Humans Apart" [29, 28, 27, 15, 9]. The idea is to introduce a difficult AI problem so that either the purpose of distinguishing bots and legitimate users is served, or that an AI breakthrough is achieved [29, 28]. The robustness of CAPTCHA systems relies not on the secrecy of the database, but on the intrinsic difficulty of the problem. The difficulty of solving a CAPTCHA problem for a bot and for a human often increases in similar curves. As CAPTCHA systems are rarely stand-alone and are often integrated as an auxiliary part for applications such as online registration, it is unrealistic to ask for the user's concentration for longer than a few seconds. Hence a complicated challenge requiring the humans to devote more time would make it unrealistic to be deployed on real world systems.

Identifying distorted letters, answering questions based on images are a few techniques that are in use to defeat bots, with the former being the most widespread. However with the increasing advances in the field of computer vision, bots have been known to break text CAPTCHAs using techniques such as OCR (Optical Character Recognition) and segmentation [30, 26, 16, 2, 19]. Increasing the complexity of the text-based systems by introducing more noise and distortion to make the challenge difficult for bots also makes them less user friendly and less usable to normal users.

Image-based systems were then proposed to increase the usability of CAPTCHA systems [20, 3, 17, 23, 18, 7, 25, 32]. However, many current state-of-the-art image-based systems such as Asirra [20] suffer from the lack of flexibility and adaptability. Assira challenges focus on image recognition only, requiring the user to identify all cats among a series of images of cats and dogs. Specialized attacks using machine learning techniques have achieved a high rate of success against systems like Asirra, as shown by Golle [22]. Moreover the inherent choice presented to the bot is always binary (an image is either a cat or a dog), making it more susceptible to template fitting attacks, which will be further discussed in Section 4.2. We propose SEMAGE, a novel image-based CAPTCHA system, which has a two-factor model requiring the user to recognize the image and identify images that share a semantic relationship.

The introduction of semantic correlation makes SEMAGE more robust from similar machine learning attacks. Other image-based systems like ESP-PIX [3] and SQ-PIX [7] are language dependent and have usability concerns. We survey more CAPTCHA systems and their limitations in Section 2.

In this paper, we propose SEMAGE (**Se**mantically **Ma**tching Ima**ge**s), a two-factor CAPTCHA system. In SEMAGE, we present the user with a set of candidate images, out of which a subset of them would be semantically related. The challenge for the user is to identify the semantically related images based on the context defined by the system. Note that the images in the correct set need not be images of the same object, a set of semantically related images may be images of entities with different physical attributes but sharing the same meaning in the defined context. Consider for example the user being asked to identify similar images with the context being similar images should have the same origin, the candidate set could contain images such as a wooden log, a wooden chair, a matchstick, an electronic item, an animal, and a human, with the chair, matchstick and log being the similar set.

The challenge in solving a SEMAGE CAPTCHA system is twofold: (1) a user has to figure out the content of the individual images, i.e., image recognition, (2) and understand the semantic relationships between them and correctly identify the matching images. This challenge solving ability comes naturally to humans as humans automatically employ their cognitive ability and common sense without even realizing the inherent difficulty of the task. The same challenge for a bot would require both understanding images and identifying relationships between them, constituting a difficult AI problem. Our two-factor design aims at increasing the difficulty level for a bot and improving usability for humans, without sacrificing the robustness of the system.

What makes SEMAGE novel is the idea of presenting the user with a two-factor challenge of "identifying images with similar semantics under the given context". The idea of choosing images exhibiting semantic similarity has a much broader scope than simple selection of images of animals of the same species (cats in the case of Assira). This feature differentiates SEMAGE from other state-of-the-art image-based CAPTCHAs that only require the user to solve the first level, which is image recognition. Computers are hard to comprehend and identify the semantic content of an image, making SEMAGE very robust to bots. We present and discuss what semantic similarity entails in Section 3.

We also implement one very simplified sample instance of SEMAGE using real and cartoon images of animals. The relationship query asks the user to pick up images (real and cartoon) of the same species. This particular implementation has two immediate benefits: (1) Adds fun factor for the user without adding burden on the recognition part since a human can easily make a connection between a real image of an animal and a cartoon image; (2) Scales up the difficulty level for bots as the cartoon images need not even resemble the real physical attributes of the animal. Moreover, SEMAGE provides an easy-to-operate interface to indicate correct answers making it an ideal choice for touch-based systems and smart-phones where typing is more difficult. A sample simplified SEMAGE challenge is shown in Figure 1 which illustrates the idea. A human can easily identify the images marked in a circle as similar but a bot would not be able to relate the real and cartoon images due to difference in shape and texture. Note that this is just one way of creating a SEMAGE challenge. Any other *semantic relationship* can be used as the identifying factor apart from our particular simplified implementation.

The main contributions of this paper are as follows:

- We propose SEMAGE, a new image-based two-factor CAPTCHA that has several unique features. The design of a SEMAGE allows easy tuning of the security level and usability level depending on the nature and popularity of the website. The images of the SEMAGE challenges can vary to suit the needs of different websites. In fact in most cases given a labeled database it is very easy and intuitive to come up with a definition "semantic relationship" and SEMAGE implementation. We also provide an in-depth security analysis and show how SEMAGE is more robust to many attacks than existing systems.
- We further conduct a large-scale user study with 174 participants using a simple sample SEMAGE implementation. We compare our system with state-of-the-art text-based CAPTCHA system reCAPTCHA [6] and image-based system Asirra [20] on the metrics of usability and fun factor. As discussed in details in Section 5, results show that our system is easy to use and participants reported a high level of 'fun' factor.

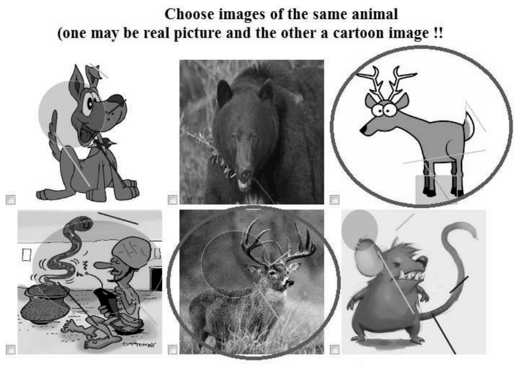

Choose images of the same animal
(one may be real picture and the other a cartoon image !!

Figure 1: Sample SEMAGE challenge; the encircled images are similar.

2. BACKGROUND

CAPTCHA systems, text-based in particular, have been in widespread use as the first line of defense against bots on the web. Recently, with the improvements in computer vision technology, text-based systems have become susceptible to bot attacks with a high success rate [30, 26, 16, 2, 19, 13]. Hence a lot of work has proposed alternate CAPTCHA systems such as image-based [20, 3, 17, 23, 18, 7, 25, 32] and audio-based systems [14, 10, 1, 21].

2.1 Text-based Systems

Generally, text-based CAPTCHA systems ask the user to discern letters or numbers. GIMPY is one classic example [4]. Attacks on text-based systems mostly employ OCR (optical character recognition) algorithms. These algorithms first segment the images into small blocks each containing only one letter, and use pattern recognition algorithms to match the letters in each block to standard letter template features [30, 26, 16]. The later task is considered a well solved AI problem. In counter-attack to these algorithms, text-based CAPTCHA systems employ the following techniques to enhance robustness [15, 19]:

- Adding noises in the form of scattered lines and dots to the background to counter-attack segmentation algorithms.
- Characters are connected or overlapped so that attacking algorithms cannot correctly segment image into correct blocks.
- Characters are twisted to increase difficulty in character recog-

nition.

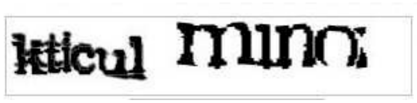


Figure 2: A text-based CAPTCHA example

However, all the above techniques increase the difficulty level for humans too. Connecting characters together makes the task harder for humans. For example, when the character 'r' and 'n' are connected, it looks like the character 'm'. Twisted characters not only gnaw on user's nerves, but also are sometimes impossible to identify correctly. Figure 2 shows one such difficult-to-solve text-based challenge.

Text-base system faces one inevitable situation: humans find the CAPTCHA challenge unpleasant as CAPTCHA gets more complicated. This is probably why popular websites such as MSN hotmail opted for simple and clean CAPTCHA , which could be attacked with a success rate over 80% [30]. Some systems use distinctive color for each character and add colored background using non-text colors, both of these additions can be easily removed by an automated program, which add no more difficulty for the bot [31].

Popular systems such as 'reCAPTCHA' [6] use dictionary words that are labeled as unrecognizable by real automatic OCR programs running on real tasks of digitizing books, and evaluate correctness by other user's input. However, reCAPTCHA also suffers from decreased usability and user satisfaction due to the high distortion and noise in the challenge.

2.2 Audio-based Systems

Audio-based CAPTCHA systems [1, 14, 10, 21] remedy the fact that visual CAPTCHA systems are not accessible to visually-impaired people. In a typical audio CAPTCHA system, letters or digits are presented in randomly spaced intervals, in the form of audio pronunciation. To make the test more robust against bots, background noises are added to the audio files. These systems are highly dependent on the audio hardware and the user only has a certain small amount of time to identify each character. In some sense, audio CAPTCHA systems can be considered as the acoustic version of text-based systems. Although the visual cues are replaced with acoustic cues and the algorithms vary, the underlying idea of attacking is the same - features are extracted and classified to recognize the letters [12]. The difficulty curve for bot and humans are similar. Thus audio CAPTCHA systems provided neither more user-friendly interface for visually accessible users, nor more robustness against bots [11].

2.3 Image-based CAPTCHA systems

Image-based CAPTCHA systems emerged in efforts to replace text-based CAPTCHA systems which were growing more complex for humans to solve easily. Security is not the only concern in a good CAPTCHA design. All CAPTCHA systems are a form of HIP (Human Interactional Proofs) and require the users involvement. This also makes usability a key issue in CAPTCHA design. Tygar et. al. [17] propose the following requirements for a good CAPTCHA system:

- The task should be easy for humans.
- The task should be difficult for computer algorithms.
- The database should be easy to implement and evaluate.

The general basis of image-based CAPTCHA is that images contain more information than texts. It is intuitive for human to catch visual cues but hard for AI algorithms to do visual recognition. ESP-PIX [3] presents a set of images and asks the user to choose a word from a list of words that describes all images. This approach suffers from two drawbacks, i.e., it still depends on text to convey meaning and since all words are written in English, and the user's success depends on his/her proficiency in English (or any other particular language it migrates to). It is not only language dependent but also hard to operate; a user needs to scan through the whole list of words to find the most proper answer. SQ-PIX [7] also presents user with an image set, but asks the user to select an image of a given object name, and also trace the object in the image. This is also language dependent and the act of tracing around an object with a pointer operated from a hand-held device like a mouse cannot be assumed to be easy for all users.

Google's image CAPTCHA "what's up" [23] asks the user to adjust the orientation of an image. This system is language independent, but the adjustment requires a lot of attention and subtle mouse (or other hardware) movement. Some images also have ambiguity as it can be correctly oriented in multiple ways.

Microsoft's Asirra [20] utilizes an existing database on petfinder.com and presents the user with images of cats and dogs and asks the user to identify all images of cats out of 12 pets. This platform is language independent, and requires user to scan through 12 images and click 6 times on average to be correct. Figure 3 shows a sample Assira challenge.

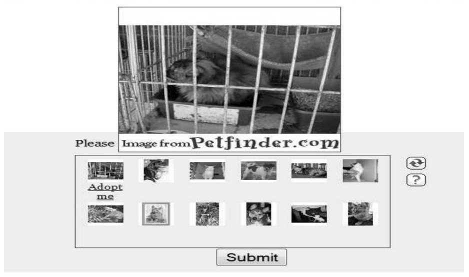


Figure 3: An Assira challenge: A user is always required to select all cats from images of cats and dogs.

Asirra partners with petfinder.com and gets access to their huge database of cats and dogs. But the inherent difficulty for the bot boils down to only classifying each image in either of the two classes: cats and dogs. This makes Assira more vulnerable to machine learning attacks [22]. SEMAGE on the other hand has a two-factor design where in order to pass a challenge the user is required to recognize each image and then understand and identify the semantic relationship between a subset of them. Assira only requires the user to solve the first level (i.e., image recognition). Utilizing the semantic correlation between images to create more secure and user-friendly challenges makes SEMAGE more robust.

3. SEMAGE DESIGN

We propose SEMAGE, "**SE**mantically **MA**tching Ima**GE**s", a novel image-based two-factor CAPTCHA system which is built upon the idea of semantic relationship between images. The use of semantic meaning of a query has already been applied in other fields like web search [24]. We formulate definitions for semantic similarity of images and design a system that uses these concepts to develop a user-friendly and robust CAPTCHA system.

3.1 Intuitive Idea

All image-based CAPTCHA systems have two main components: a database of images and a "concept" which uses the database to create challenges. The inherent concept may be as simple as PIX [8] which displays different images of the same object from the database and asks the users to assign an appropriate label or a complex one like Cortcha [32] which uses the database to create inpainted and candidate images and asks the users to place the correct candidate image in the inpainted image.

The idea behind SEMAGE is to use semantic relationships among images as the concept and keep the task of the user to simply identify the semantically similar/related images. The semantic relationship is a concrete description which would bind the similar images. The freedom of choosing the semantic relationship for one's application and database gives it the much required customization flexibility. For example, for an electronic e-commerce site, SEMAGE challenge could be formed from the images of the products (an ipod, a zune, tv, heater, refrigerator etc) where the concept would be to ask the users to choose products which do the same thing (ipod and zune in this case, both portable music devices).

SEMAGE presents a set of candidate images with a subset of them sharing an implicit connection or relationship with each other. The challenge for the users is to correctly identify all images in the semantically related subset.

3.2 Defining the Semantic Relationship

We now present the conditions for choosing the "semantically similar" relationship which forms the 'concept' for challenge creation. A "semantic label" could be a term or a relationship which identifies/labels the object. Semantic labels can be directly used to label the database for challenge creation. Let $SL(x)$ denote the function that returns the semantic label of an object x. We consider two images to be "Semantically Matching" if they satisfy any of the following conditions:

- Condition I: if both images can be identified with the same semantic label. Given two images A and B, they are said to be semantically related if $SL(A) = SL(B)$. For example, an image of a computer and a television set can be defined with a semantic label(SL) 'electronics'.
- Condition II: both images can be classified under the same semantic label. Given two images A and B, they are semantically related if $\exists T\, s.t.\, SL(A) \subset T \,\&\, SL(B) \subset T$, where T denotes some semantic label. For example an image of a lion and a deer can be classified under the semantic label 'four legged animals'. Similarly, an image of a television set and a computer can be classified under the semantic label 'electronics'.
- Condition III: when both images put together they express a uniquely identifiable concept. Given two images A and B and some semantic label C that denotes a set of requirements, A and B are said to be semantically matching if $\{A \cup B\} \models C$ where "$\models$" denotes that the left hand side satisfies the requirements of right hand side. For example, an image of a printer and paper can be defined with a identifiable concept 'printing' which becomes the semantic label.

The requirements for a "semantic relationship" gets more generic and the semantic correlation increases as we move from Condition I to III. In order to form a SEMAGE challenge, the images have to be chosen such that only one subset meets any one of the above conditions with preference given to the least generic label. That is, if a set of images contain images that satisfy more than one of the above conditions, the least generic matching is the solution required to pass the challenge. Thus, given a set of images where a small subset of images is of fishes and the rest of the images are of other unique animals, the solution to the challenge would be selecting all images of fishes.

The mechanism may seem complicated but as we show below, a system designed to create challenges where all solutions satisfy only one chosen condition is relatively easy to implement. Also the user study in Section 5 supports our claim that such a system is intuitive and easy for the normal user to solve. The important thing after one has decided upon the "semantic relationship" is to label the images accordingly. We discuss database generation in Section 3.4.

3.3 Challenge Creation

We develop a simple algorithm to create SEMAGE challenges. First we present the definitions and requirements of the involved parameters as follows.

Let n be the number of images in the challenge and m be the number of similar/related images. Let U be the superset of all image sets in the database. Each challenge set is denoted as S where $|S| = n$. There exists a 'semantically similar' subset of images R such that every image in R has the same semantic label, i.e., $\forall\, r_i,\, r_j \in R,\, SL(r_i) = SL(r_j) \,\&\, |R| = m$. A set of images D with $|D| = n - m$, and each image in D has a different semantic label than R. Also $\forall\, d_i,\, d_j \in D,\, SL(d_i) \neq SL(d_j) \neq SL(R)$. This ensures that all the images in the subset D have a different semantic label so that the images in subset R remain the unambiguous semantically related set. Now each challenge set becomes $S = R \cup D$.

We now present a simple algorithm to implement the challenge set as shown in Algorithm 1 . The database consists of a collection of semantically labeled images. The algorithm starts with empty sets R and D. We then pick a semantic label at random from the database and populate R with images having the picked semantic label. Then we populate D with images such that each image has a different semantic label than any of the images chosen previously in D and R. The number of images in the R and D depends on the values of n and m and is customizable. The images in set $R\, and\, D$ are then presented in a random tabular order to the user.

Algorithm 1 : An algorithm to generate SEMAGE challenges from a labeled database

```
$R \leftarrow \phi$
$D \leftarrow \phi$
$A \leftarrow$ Pick an Semantic label at random
while $|R| \neq m$ do
    $X \leftarrow$ (pick a unique image with label $A$)
    $R = R \cup X$
end while
$Y \leftarrow \phi$
while $|D| \neq (n - m)$ do
    $Z \leftarrow$ Pick a label at random which is not $A \cup Y$
    $Y \leftarrow Y \cup Z$
    $D = D \cup$ (pick a unique image with label $Z$)
end while
$S \leftarrow R \cup D$
Randomize(S)
```

3.4 Database

Populating the database is a major issues with all image-based systems. Unlike text CAPTCHAs which can use any random combination of characters in the challenge creation, images in SEMAGE owing to the requirement of semantic similarity have to be carefully selected. One may always use freely available image search services like google image search to find relevant images. For our implementation, we developed a semi-automated mechanism that populates the database by crawling the Internet. One can also consider taking frames from movies and short videos. Both of the above approaches can be considered as semi-automatic and require some manual work to weed out irrelevant images. The drawback of such methods is that an attacker can venture to spend enough time and manual work to reproduce the whole database.

SEMAGE, however, due to its inherent design offers an way of database creation for web sites, such as e-commerce sites, which already have a image database. Web vendors in e-commerce usually have multiple images of the same product (such as pictures from different angles), multiple styles of the same product (same product of different color, size, packages), and multiple products of the same category. Images are tagged with the product information, and product info is categorized into different classes. Multiple relations can be established among these images and used as the 'semantic context'. With the abundance of existing tagging information, we can implement the 'challenge creation' algorithm by adding simple logical changes. Furthermore, some databases actually have implemented more sophisticated relations such as 'similar products' as a recommendation for users when they browse certain products, thus more sophisticated 'semantic relationships' can be formed based on such information. Using these images not only adds to the security of the database, but also serves as a good form of advertisement.

4. SEMAGE ANALYSIS

4.1 Design Analysis

4.1.1 Usability

Usability with security is the primary focus of SEMAGE. The images contain content that cognitively make sense to the users, and are easy to discern. By drawing on human's vast storage of common-sense knowledge, our design helps user spend minimum effort solving the challenge. Moreover, it fits the way a human thinks - it is natural for humans at first sight to see what an image is about, much better than dealing with any details (orientation, certain feature image, etc.). Establishing relationships among objects is another ability humans are natural at, and humans almost automatically dissolve any ambiguity they need to resolve. For example, if a red car is presented with other colored cars, human immediately notice the color difference. However, if the same red car is presented with red buckets, red clothes etc. humans notice the difference in object category. For a computer, both of the steps pose a difficult AI problem. It first needs to do image recognition to determine what the image contains, and tag the image in a pre-determined category. To solve the 'relationship' answer, the computer would not only need vast correctly labeled database, but also complex AI intuition. This creates a great gap in the difficulty level for humans and bots.

In addition, SEMAGE provides an easy-to-operate interface for users to indicate correct answers. Only a few mouse clicks is required to pick up the correct images, this makes SEMAGE to be a good choice of touch-based systems and smart-phones where typing is more difficult. This is much easier than tracing an outline of objects (as in SQ-PIX [7]) and typing in letters from a keyboard, especially on mobile devices.

4.1.2 Language Independence

Our design utilizes the fact that a picture transcends the boundaries of languages. Some CAPTCHA systems also use semantic clues, such as ESP-PIX [3]. However it asks the user to find the right word among a list of English words that describes the content of the image. This limits the audience to people with decent proficiency in the language. Our design is language independent and can be used by people across the world. This is especially beneficial for people who are not comfortable using English as a daily language.

4.1.3 Customization Flexibility

Our design offers several ways to customize the challenge on content, security level and usability level. The image database can be customized to suit the needs and style of the hosting website. For example, for special interest groups, the database can be objects of the theme of the group, such as movie screenshots for a movie rental site or specific products for an e-commerce site. This provides possibility of advertisement of content or fun in the traditionally boring test of CAPTCHA.

It is also easy for web administrators to customize on the security level. The administrator can decide on the size of the candidate image pool, and the size of the correct answer set. For a scheme that present n candidate images and ask the user to pick up k matching images, the success rate of random guessing is $1/C(n, k)$. The increase of the size of answer set does not necessarily decrease the chance of success of a random guessing success when n is small, but as n increases, the probability of a random guess attack goes down. As for the user experience, the time users spent on the CAPTCHA task increases as the size of candidate image pool increase, but the effect of an increased size of answer set on users time is not obvious. We think the optimum choice of n and k might depends on particular content of the images used, and a specialized user study can be conducted if such data is desired.

4.2 Security Analysis

We consider an adversary model wherein a bot has access to the unlabeled and uncategorized database of images from which we form our challenges. It is to be noted that given ample time and resources some of the attacks discussed below could succeed but taking a long time defeats the primary purpose of the bot. Our goal as in any CAPCTHA system is to make current attacks as difficult as possible, so that any successful attack would need a major step forward in technology. We now identify and analyze possible ways of attacks against our system and how it fares against them.

4.2.1 Attacks using machine learning techniques

Similar techniques used to attack Asirra [22] could be used to attack our system too. The attack on Assira was an attack on the first level of our model namely simple "image recognition". In essence, attackers try to get a certain number of correctly labeled images, and train on several different classifiers, either based on color information or texture information. However, solving a SEMAGE challenge not only requires image recognition but also identifying the "semantic relationship". The identification of "semantic relationship" among images is an unsolved AI problem. Moreover, even if the semantic correlation is weak and the semantic label is just the object name, SEMAGE accommodates much more object classes than Asirra (which had only 2), and the attacker will need to build many more types of classifiers accordingly.

Now let us consider a very simple example of "semantic relationship", e.g., "real and cartoon" images of the same animal (as used in Section 5). The color and texture data between a cartoon specie and real animal specie varies much more than in between cartoons and real animals, as illustrated in Figure 4. While attackers might attempt to train classifier of real animal and cartoon animal independently, the performance decreases as the number of classifiers increase which could be very complex. Thus the success rate of attacks using this sort of algorithm is likely to be very low.

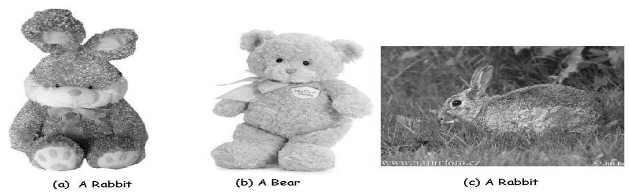


Figure 4: Example limitations of the texture-based machine learning attack; (a) shares more commonality with (b) than with (c) , while (a) and (c) are of the same type (rabbit).

Attacks using template fitting techniques: In image recognition, one developed area is to fit objects into (visual) feature templates. For example, a chair can be identified if given the template of 'four legs and a horizontal top'. Accordingly, for a rabbit, the feature should probably be 'upwards pointing long ears'. However, it is much harder to define 'long' than 'upwards'. A deer, with pointy upward ears would be classified into the 'rabbit' template. Furthermore, not all objects have such uniquely identifiable simple feature.

4.2.2 Random guess attack

For a SEMAGE scheme that presents n candidate images and asks the user to select k matching images, the success rate of random guessing is $1/C(n, k)$. As shown in Figure 5, choosing a low value of n and k could make the system more vulnerable to random guess attacks. On the other hand a low n, k makes the system more user friendly and less frustrating for the user. Our implementation for the user study uses low n, k values making it more susceptible to random guess attacks. In case of a low n, k system, multiple rounds of SEMAGE could constitute one challenge; such technique is already in use in current systems such as reCAPCTHA. By choosing a relatively low n, k value, we sacrifice a bit of security against random guess attacks for usability. We do so because we can make up for the relatively high susceptibility of SEMAGE to random guess attacks and deter brute force attackers by enhancing SEMAGE with Token Buckets [20] system. Assira needs more images in each challenge set to be secure because of the limited set of differentiating classes of objects (two to be precise, just cats and dogs) whereas there can be theoretically thousands of differentiating classes in our SEMAGE implementation. The added security provided by SEMAGE's two-factor design allows us to use a low n, k system without sacrificing security much.

A SEMAGE system could also be complemented with other techniques such as the Partial Credit Algorithm in [20], which would allow a large n, k and an 'almost right' answer can be defined as missing one image in the answer set. Token buckets [20] can also be implemented to prevent brute-force attackers from making a number of continuous random guess attacks.

4.2.3 Attack using the static image name in source

If the source code of the HTML page hosting the challenge uses image names, an attacker could potentially use those names to identify similar images. However, this sort of attack is easily defeated by randomizing the images name in the source. In our system implementation, names of the images in the challenge are in no way exposed to the user. The image names in the html source is randomized when sent to the user.

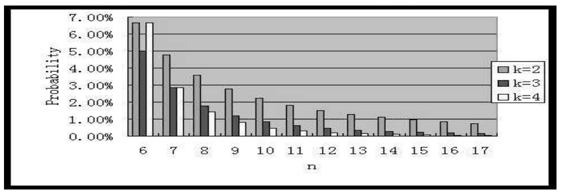


Figure 5: Random guess attack success rate with respect to k and n

4.2.4 Attack by creating an attack database using the general relationships used in the system

The attacker might manually identify the general "semantic relationship" used in the system and then search and build an image repository to create an attack database. Using the labeled images of the attack database, a brute force search against the candidate set might yield him a correct 'similar' set. However comparing each image of the challenge with all the images in the attacker's image archive would take lots of time and resources than what would constitute a feasible attack; also this might exceed the maximum time allowed to take a challenge.

4.2.5 Attack by mining Textual description of images

Potentially an attacker could use systems such as google's goggle[1], an image based search system, to uncover textual descriptions of the candidate image set and then use the textual descriptions to identify relationships among images. We argue that first of all image recognition or search is still not mature enough for now (very hard problem for unknown images). In addition, identifying relationships among objects even with textual descriptions is a complex AI problem to solve, especially since the correct similar images depend on the semantic context. Such an attack would potentially defeat most present image-based systems such as Assira, PIX, SQ-PIX, but because of the two level design of SEMAGE, the bot would still need to understand and identify the semantic correlation. Having a textual description only possibly solves the problem of image recognition. There may exist images with overlapping descriptions but are not a part of the 'semantic similar' image set in the context. Consider for example a candidate image set wherein the context is identifying 'four legged' animals among images of insect, deer, lion, human, electronics item and other unrelated objects. Now even with accompanying textual descriptions such a relationship is hard for a bot to find and relate to lion and deer.

5. EVALUATION

We conducted a large-scale user study to evaluate the usability of SEMAGE as compared to Assira and reCAPTCHA. For this purpose, we firstly built a website which would present the users with sample SEMAGE challenges.

5.1 Sample Implementation of SEMAGE

In our sample implementation, each challenge consists of a set of images (the number of images is configurable) where a subset of images would share a distinct relationship/feature with each other. The images are furthermore randomly distorted by introducing noise and changing the texture. Our implementation was carried out in PHP with MySQL being used as the database. Figure 6 gives a high level design of the implementation.

[1]`http://www.google.com/mobile/goggles/`

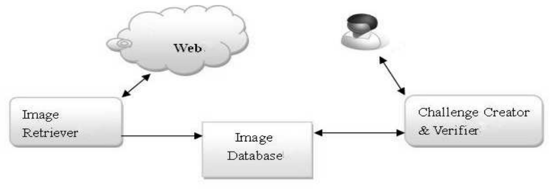


Figure 6: Overall Implementation Illustration

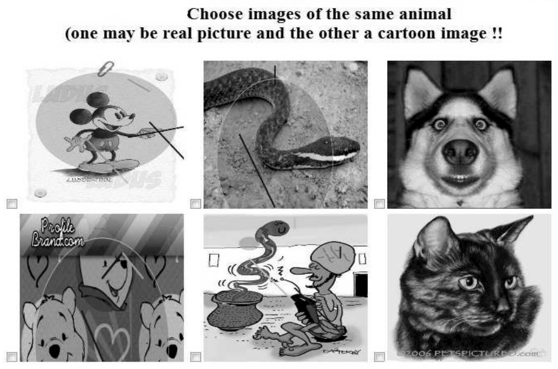

Choose images of the same animal
(one may be real picture and the other a cartoon image !!

Figure 7: Screenshot of sample SEMAGE implementation with Image 2 and 5 being similar, both snakes.

Choosing the "semantic relationship": In our particular implementation, the challenge set consists of real and cartoon images of animals with the relationship defining the 'similar' subset being "real and cartoon images" of the same animal. The advantages of choosing the 'real and cartoon' relationship to define "semantic relationship" between images are as follows:

- The relationship between real and cartoon images of the same animal in most cases is subtle and variable. The reason is that the animals may completely differ in visual characteristics such as size, shape and outline in real and cartoon representations.
- Humans with inherent capability to relate visibly dissimilar objects would be able to pass the challenge easily whereas the current state-of-the-art bots cannot. We test this assumption of ours in the user study we conduct, discussed in details in Section 5.
- Generating a large database is easier. A simple search for an animal on images.google.com yields millions of entries, hence we have a fast and easy way to build up a large database.

Figure 7 shows a sample SEMAGE challenge of our simple implementation. The total number of images in one challenge is six with the "semantically similar" set of two images, one a real image and the other a cartoon image of the same animal.

Database Generation: The first step for SEMAGE implementation after defining the semantic relationship between the "similar" images is database generation. An image search and download tool was implemented shown as Image Retriever in Figure 6, which searches and downloads the required images from the web. The tool would take in the search keywords (to search for real or cartoon images of the animals), image dimensions, and number of images to download and the label tags. It then automatically downloads the images and stores in the database. A simple search for an animal on images.google.com yields millions of entries, hence we have a fast and easy way to build up a large database. In reality, since the automated search does not always yield relevant results, we manually weed out the irrelevant images from the collection.

Dynamic Noise Addition: To make machine learning attacks based on image classifiers difficult, we randomly introduce noise in the images of the challenge set at each challenge creation phase. We introduce noise in the form of random shapes and color scale alteration in the image with the help of the ImageMagick library [5]. The position of inserting the random shapes varies from the center of the images to its edges. Also scale of color adjustment is also randomly varied to prevent the bot classifiers from easily weeding out the noise. Such random noise introduction makes sure that each image appears with different noise levels. Figure 8 shows a SEMAGE challenge after the introduction of noise.

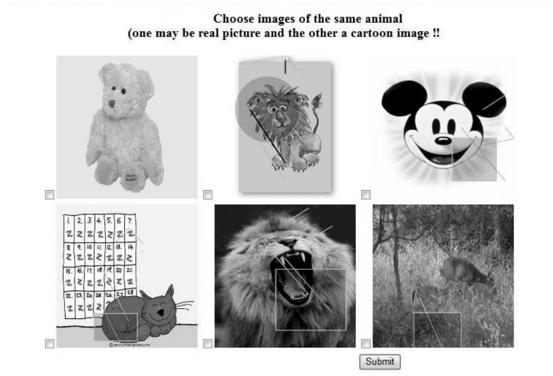

Choose images of the same animal
(one may be real picture and the other a cartoon image !!
Submit

Figure 8: Example of noise addition in our implementation. Here we can clearly see noise but still identify Image 2 and 5 being similar, both lions. The changes in color scale are not visible due to the black and white nature of images.

Interface: As shown in Figures 8 and 7, each challenge appears as a tabular strip of images. The title of the tabular strip presents the challenge and then the user needs to click on the similar images and press submit to send the response to the server for verification. We experimented with different layouts, e.g., the images being apart from one another, images in a single straight strip, and found that it is much easier to identify similar images if they are bunched together in a tabular format.

5.2 User Study Methodology

A comprehensive IRB approved user study was then conducted to gather data about how user-friendly SEMAGE is, which is one of the most essential criterion for a CAPTCHA to be deployed in real systems. We also incorporated reCAPTCHA, a text-based system and Asirra, an image-based system from Microsoft in the user study to carry out a comparative analysis. Both Asirra and ReCAPTCHA are available as a free web service allowing us to easily integrate them in our study. The volunteers took the study remotely and were given a brief 1-page pictorial description of what they need to do to pass a challenge for all the systems. We logged the time taken to complete each challenge as the difference in time between when the test first appears on the screen and the time user clicks on the 'submit' button to submit his attempt. The users were let known of whether they passed or failed the previous challenge before presenting a new one.

A total of 174 volunteers took the study and the population was a mix of graduate and under-graduate students. The subject pool was diverse with most of the users from a non-computer science discipline, with a mix of native and non-native English speakers. The subject pool consisted of 66 females and 108 males. The subject pool were in no way made aware of the fact that SEMAGE is our system. We collected the time taken by each user to complete a challenge for each of the system as described earlier. We monitor the time taken for all attempts irrespective of whether it was successful or not. We also collected numbers of successful and failed attempts to solve a challenge.

5.3 User Study Layout

The user study was carried out via a website with the following sections:

- An initial questionnaire asking the users to rate their familiarity with CAPTCHAs, proficiency in English language and other demographic questions such as sex and age range.
- A 1-page pictorial description of EMAGE, Assira and reCAPTCHA, showing users how to solve each challenge.
- 5 different challenges from SEMAGE.
- 5 different challenges from Asirra.
- 5 different challenges from ReCAPTCHA.
- A final short questionnaire asking users to rate SEMAGE for fun factor and ease of use as compared to Assira.

We believe a pictorial description of each of the systems was necessary for fair usage statistics on the image recognition systems. It was probably a user's first time seeing an image-based CAPCTHA whereas all the users had invariably taken a text-based challenge before. Presenting a brief description of what they need to do to pass a challenge would prepare them with necessary basic information of each system and allow us to collect fair usage data. The study took an average of 8.7 minutes to complete.

We divide the usability evaluation in different sections presented below according to the following metrics:

- How fast can a user complete a challenge?
- How many times does the user pass the challenge successfully?
- Does the user consider the system to be fun and easy?

5.4 Timing Statistics

As shown in Table 1, users complete text-based and SEMAGE challenges faster than Asirra. Each user takes an average of 6 seconds more to complete an Assira challenge.

	Semage	**Asirra**	**recaptcha**
Time Taken in seconds	11.64	17.35	11.05

Table 1: Average Time taken per challenge for each of the systems (in seconds)

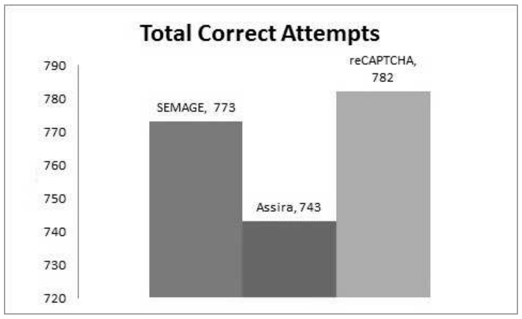
The distribution plots in Figures 9 show that most of the users of SEMAGE finished each challenge in about 11.647 seconds or less, whereas this number is comparatively higher for Asirra with most of the users taking around 17.355 seconds. Consistency and uniformity in majority of the data points of the plots show that the timing average was not largely affected by some isolated outlier cases and it represents the general behavior of the users.

We notice that the average time taken by the users to solve a challenge from SEMAGE is almost the same as that of reCAPTCHA. This is actually surprising. We expected that solving a SEMAGE challenge is much slower than solving a reCAPTCHA challenge because text-based CAPTCHAs have been widely in use for a long time and users have gotten used to them whereas users were seeing our system for the very first time. This encouraging fact suggests that SEMAGE is pretty user-friendly and easy to use.

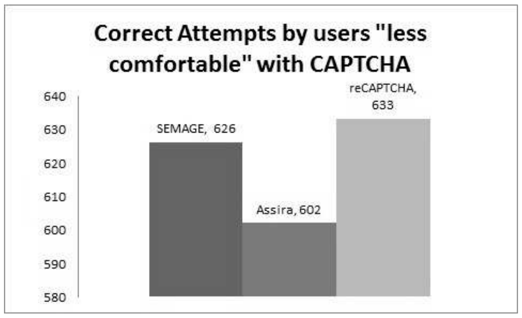
We concede that an Assira challenge consists of more images than a SEMAGE challenge leading to more time spent in completing each challenge. However, Assira needs more images in each challenge set to be secure because of the limited set of differentiating classes of objects (two to be precise, cats and dogs) whereas there can be theoretically thousands of differentiating classes in our SEMAGE implementation. Moreover, presence of just two differentiating given classes should have made the challenge easier for humans as they simply need to place each image in one of the two categories. SEMAGE on the other hand requires the user to relate two or more images, making it potentially more time consuming. However the timing data clearly shows that taking SEMAGE challenges is easier than it seems because of the natural cognitive ability of humans.

5.5 Accuracy Statistics

Simply speaking, the total number of correct attempts for SEMAGE is higher than Asirra, indicating that users are able to correctly solve more challenges of SEMAGE. Figure 10(a) shows a graphical representation of the difference in correct attempts between Assira and SEMAGE. We had also asked the users to rate their familiarity and comfort level with CAPTCHAs on the scale of 1 to 5 (with 5 being very comfortable) in the initial questionnaire. As we see in Figure 10(b), the participants who voluntarily identified themselves as 'less comfortable' (rated 3 or less) with CAPTCHA systems in general also show high accuracy with SEMAGE and reCAPTCHA than with Asirra.



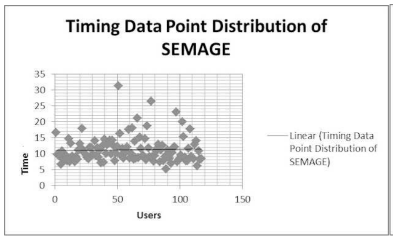
(a) Total correct attempts out of 815 attempts



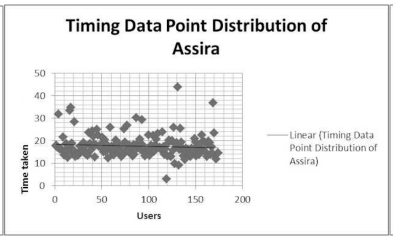
(b) Total correct attempts from 132 users who rated themselves as less comfortable with CAPTCHA

Figure 10: Accuracy achieved on individual systems

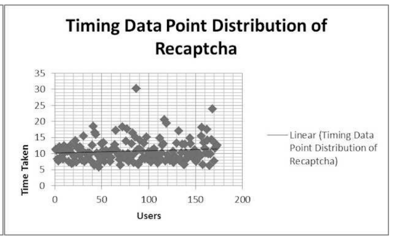
In order for the system to be deployed in the real world, it should have a high 'Correct Attempts ratio' for humans. The 'Correct Attempts ratio' (C.A.R) is simply the number of correct attempts di-



(a) SEMAGE Timing



(b) Assira Timing



(c) reCAPTCHA Timing

Figure 9: Timing Distribution of each system for all users

vided by the total attempts. It signifies how many times a human passes the challenge. The closer the ratio is to 1, the better the system is in terms of usability.

The user study data shows that our system has a higher C.A.R (0.94) than Asirra (0.91). Users had been familiar with text-based CAPTCHA systems, so we expected them to do very well in the reCAPCTHA system. But again, the difference between SEMAGE and the traditional text-based system is almost negligible. This along with the timing data shows that our system likely has a higher usability factor than the current state-of-the-art image-based system (Asirra).

5.6 Fun Factor and Ease of Use

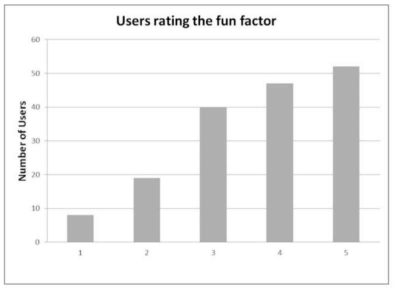
After the completion of challenges from the three systems, the users were then asked to compare and rate SEMAGE and Assira on the criterion of Fun and Easiness. There were two separate questions: one for Fun factor and the other for Easiness, which asked them to choose a rating as follows:

- 1, if they found Assira to be way more fun or easy
- 3, if they found Assira and SEMAGE to be equal on the Fun or Easiness factors
- 5, if they found SEMAGE to be way more fun or easy
- 2 or 4, if they were slightly inclined towards Assira or SEMAGE, respectively.

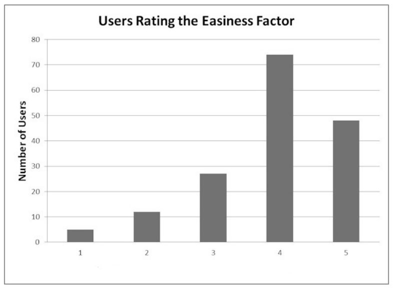
These factors gave us a more subjective indicator of usability. We can clearly see from Figure 11(a) that majority of the users (58.92 %) choose rating 4 and 5 indicating a high fun factor with SEMAGE. Only 16.07% choose rating 1 and 2 indicating Assira was better while the rest considered them to be equally fun. This clearly supports that more users found SEMAGE to be a system that was more fun to solve than Assira.

Figure 11(b) shows the rating distribution for the easiness factor. 72.61% of the users rated 4 and 5 indicating SEMAGE to be easier than Assira. Only 10.72% of the users rated 1 and 2 indicating Assira to be easier while 16.66% of the users rated 3 indicating they considered both systems to be equally easy.

These metrics as well as the timing and accuracy results shown previously clearly demonstrate that SEMAGE is a highly user-friendly CAPTCHA system.



(a) Users rating the Fun factor



(b) Users rating the Easiness

Figure 11: We asked users to comparatively rate SEMAGE and Assira on the metrics of fun and easiness. Rating 1,2 indicate Assira to be more fun and easy, rating 3 indicate both systems are equal and rating 4,5 indicate SEMAGE to be more fun and easy.

6. LIMITATIONS AND FUTURE WORK

Generating a vast and correct database is always a challenge for image-based CAPTCHA systems. In our simple SEMAGE implementation we crawl the web to automatically gather and label images. However not all images returned by the crawler were relevant, some were even objectionable. We then manually weeded out the irrelevant images. Such manual labor is time consuming and would pose a big problem when the database content is regularly updated. There can also be legal issues in directly using the crawled images.

SEMAGE by the virtue of its design though, does not require the database to be built in such a way. Websites like e-commerce services, movie rental services can easily use the available image database with a suitable "semantic relationship". However, further work is required to create a large, correct database automatically to allow widespread deployment in real world.

In this paper we introduced the concept and technique of creating CAPTCHAs using "semantic relationships" between objects and then implemented a simple system for demonstration. Our naive implementation does *not* reach the full potential of SEMAGE and we plan to build a more robust, high semantic correlation based SEMAGE system as future work.

7. CONCLUSION

In this paper, we present SEMAGE (semantically matching images). The design of this CAPTCHA presents a set of candidate images and asks users to choose a set of images that fit a certain relation. The challenge is layered in that both knowledge about semantic meaning of images and relationship between the subjects of images is required. The challenge comes naturally to humans as it incorporates light-weight visual and cognitive task. However, the layering scheme provides double protection against bot attacks. It is easy to understand and the interaction interface is simple and efficient. CAPTCHA systems constantly seek an optimum trade-off point on security and usability. SEMAGE provides great room for customization by the website administrators. They can customize the number of candidate images and semantically similar images in the challenges to adjust the usability and security level according to the need of particular websites. Moreover SEMAGE can be targeted towards touch-based smart-phones and devices where typing to solve a text-based CAPCTHA is difficult. Website administrators can also determine the content of the image database and cater towards their promotional needs. The database can be populated especially for SEMAGE, or adapted from existing database. E-commerce is one area where SEMAGE database can be easily built and SEMAGE can be utilized for both security and advertisement purposes.

8. REFERENCES

[1] Audio and visual captcha. `http://www.nswardh.com/shout/`.

[2] Breaking text captcha. `http://www.blackhat-seo.com/2008/how-to-break-captchas/`.

[3] Esp-pix. `http://server251.theory.cs.cmu.edu/cgi-bin/esp-pix/esp-pix`.

[4] Gimpy project. `http://www.captcha.net/captchas/gimpy/`.

[5] Imagemagick. `http://www.imagemagick.org/script/index.php`.

[6] recaptcha official site. `reCaptchaOfficialSite:http://www.google.com/reCAPTCHA`.

[7] Sq-pix. `http://server251.theory.cs.cmu.edu/cgi-bin/sq-pix`.

[8] L. v. Ahn. *Human Computation*. Ph. d. dissertation, Carnegie Mellon University, 2005.

[9] H. S. Baird and K. Popat. Human interactive proofs and document image analysis. In *Proceedings of the 5th International Workshop on Document Analysis Systems V*, DAS '02, pages 507–518, London, UK, 2002. Springer-Verlag.

[10] J. P. Bigham and A. C. Cavender. Evaluating existing audio captchas and an interface optimized for non-visual use. In *Proceedings of the 27th international conference on Human factors in computing systems*, CHI '09, pages 1829–1838, New York, NY, USA, 2009. ACM.

[11] E. Bursztein, R. Bauxis, H. Paskov, D. Perito, C. Fabry, and J. C. Mitchell. The failure of noise-based non-continuous audio captchas. In *Proceedings of 2011 IEEE Symposium on Security and Privacy (Oakland'11)*, 2011.

[12] E. Bursztein, R. Beauxis, H. S. Paskov, D. Perito, C. Fabry, and J. C. Mitchell. The failure of noise-based non-continuous audio captchas. In *Proceedings of the 2011 IEEE Symposium on Security and Privacy*. IEEE Computer Society, 2011.

[13] E. Bursztein, S. Bethard, C. Fabry, D. Jurafsky, and J. C. Mitchell. How good are humans at solving captchas? a large scale evaluation. In *Proceedings of 2010 IEEE Symposium on Security and Privacy (Oakland'10)*, 2010.

[14] T.-Y. Chan. Using a text-to-speech synthesizer to generate a reverse turing test. *Tools with Artificial Intelligence, IEEE International Conference on*, 0:226, 2003.

[15] K. Chellapilla, K. Larson, P. Simard, and M. Czerwinski. Designing human friendly human interaction proofs (hips). In *Proceedings of the SIGCHI conference on Human factors in computing systems*, CHI '05, pages 711–720, New York, NY, USA, 2005. ACM.

[16] K. Chellapilla and P. Simard. Using machine learning to break visual human interaction proofs (hips). In *In Advances in Neural Information Processing Systems*, pages 265–272, 2005.

[17] M. Chew and J. D. Tygar. Image recognition captchas. In *In Proceedings of the 7th International Information Security Conference (ISC)*, pages 268–279, 2004.

[18] R. Datta, J. Li, and J. Z. Wang. Imagination: a robust image-based captcha generation system. In *Proceedings of the 13th annual ACM international conference on Multimedia*, MULTIMEDIA '05, pages 331–334, New York, NY, USA, 2005. ACM.

[19] A. S. El Ahmad, J. Yan, and L. Marshall. The robustness of a new captcha. In *Proceedings of the Third European Workshop on System Security*, EUROSEC '10, pages 36–41, New York, NY, USA, 2010. ACM.

[20] J. Elson, J. R. Doucerur, J. Howell, and J. Saul. Asirra: A captcha that exploits interest-aligned manual image categorization. In *Proceedings of the 14th ACM conference on Computer and communications security*, CCS '07, pages 366–374, New York, NY, USA, 2007. ACM.

[21] H. Gao, H. Liu, D. Yao, X. Liu, and U. Aickelin. An audio captcha to distinguish humans from computers. In *Proceedings of the 2010 Third International Symposium on Electronic Commerce and Security*, ISECS '10, pages 265–269, Washington, DC, USA, 2010. IEEE Computer Society.

[22] P. Golle. Machine learning attacks against the asirra captcha. In *Proceedings of the 15th ACM conference on Computer and communications security*, CCS '08, pages 535–542, New York, NY, USA, 2008. ACM.

[23] R. Gossweiler, M. Kamvar, and S. Baluja. What's up captcha?: a captcha based on image orientation. In *Proceedings of the 18th international conference on World wide web*, WWW '09, pages 841–850, New York, NY, USA, 2009. ACM.

[24] R. Guha, R. McCool, and E. Miller. Semantic search. In *Proceedings of the 12th international conference on World Wide Web*, WWW '03, pages 700–709, New York, NY, USA, 2003. ACM.

[25] P. Matthews and C. C. Zou. Scene tagging: image-based captcha using image composition and object relationships. In *Proceedings of the 5th ACM Symposium on Information, Computer and Communications Security*, ASIACCS '10, pages 345–350, New York, NY, USA, 2010. ACM.

[26] G. Mori and J. Malik. Recognizing objects in adversarial clutter—breaking a visual captcha. In *In Proceedings of the Conference on Computer Vision and Pattern Recognition*, 2003.

[27] Y. Rui and Z. Liu. Excuse but are you human? In *Proceedings of the eleventh ACM international conference on Multimedia*, MULTIMEDIA '03, pages 462–463, New York, NY, USA, 2003. ACM.

[28] L. von Ahn, M. Blum, N. J. Hopper, and J. Langford. Captcha: Using hard ai problems for security. In *In In Proceedings of Eurocrypt, Vol. 2656*, pages 294–311, 2003.

[29] L. von Ahn, M. Blum, and J. Langford. Telling humans and computers apart automatically. *Commun. ACM*, 47:56–60, February 2004.

[30] J. Yan and A. S. El Ahmad. A low-cost attack on a microsoft captcha. In *Proceedings of the 15th ACM conference on Computer and communications security*, CCS '08, pages 543–554, New York, NY, USA, 2008. ACM.

[31] J. Yan and A. S. El Ahmad. Usability of captchas or usability issues in captcha design. In *Proceedings of the 4th symposium on Usable privacy and security*, SOUPS '08, pages 44–52, New York, NY, USA, 2008. ACM.

[32] B. B. Zhu, J. Yan, Q. Li, C. Yang, J. Liu, N. Xu, M. Yi, and K. Cai. Attacks and design of image recognition captchas. In *Proceedings of the 17th ACM conference on Computer and communications security*, CCS '10, pages 187–200, New York, NY, USA, 2010. ACM.

BLOCK: A Black-box Approach for Detection of State Violation Attacks Towards Web Applications

Xiaowei Li
Department of Electrical Engineering and Computer Science
Vanderbilt University
xiaowei.li@vanderbilt.edu

Yuan Xue
Department of Electrical Engineering and Computer Science
Vanderbilt University
yuan.xue@vanderbilt.edu

ABSTRACT

State violation attacks towards web applications exploit logic flaws and allow restrictive functions and sensitive information to be accessed at inappropriate states. Since application logic flaws are specific to the intended functionality of a particular web application, it is difficult to develop a general approach that addresses state violation attacks. To date, existing approaches all require web application source code for analysis or instrumentation in order to detect state violations.

In this paper, we present BLOCK, a BLack-bOx approach for detecting state violation attaCKs. We regard the web application as a stateless system and infer the intended web application behavior model by observing the interactions between the clients and the web application. We extract a set of invariants from the web request/response sequences and their associated session variable values during its attack-free execution. The set of invariants is then used for evaluating web requests and responses at runtime. Any web request or response that violates the associated invariants is identified as a potential state violation attack. We develop a system prototype based on the WebScarab proxy and evaluate our detection system using a set of real-world web applications. The experiment results demonstrate that our approach is effective at detecting state violation attacks and incurs acceptable performance overhead. Our approach is valuable in that it is independent of the web application source code and can easily scale up.

Keywords: black-box approach, state violation attack, web application security, invariant

1. INTRODUCTION

During the past decade, web applications have become the most prevalent way for service delivery over the Internet. As they get deeply embedded in business activities and required to support sophisticated functionalities, the design and implementation of web applications are becoming more and more complicated. The increasing popularity and complexity make web applications a primary target for hackers on the Internet. According to a recent survey [22], attacks against web applications account for 63% of all Internet exploits. These attacks are commonly classified into two classes [7]: 1) input validation attacks, which exploit the application's insufficient or erroneous sanitization of user inputs, allowing malicious code to be injected into web applications, and 2) state violation attacks, which exploit logic flaws in web applications, allowing restrictive functions and sensitive information to be accessed at inappropriate states. For example, authentication bypass attack allows the attacker to perform administrative operations over the web application.

This paper focuses on state violation attacks. While there are a large body of literatures on input validation attacks (e.g., [12, 25, 15, 11]), there have been very limited works that address state violation attacks [4, 7, 10]. The major challenge for defending against state violation attacks comes from the fact that application logic flaws are specific to the intended functionality of a particular web application. Thus it is difficult to develop a general approach that addresses state violation attacks towards different applications. Clearly, the key to approaching state violation attacks is to derive the intended behavior model of a particular web application, i.e., the specification of web application. The existing works have presented both static and dynamic techniques to infer the web application specification. For example, MiMoSA [4] analyzes the application source code to derive an intended workflow graph; Swaddler [7] establishes models of session variables for each program block of the application based on the execution traces; Waler [10] infers likely invariants of session variables at each program point during execution. Such specifications can then be either leveraged by model checking (e.g., MiMoSA, Waler) to identify vulnerabilities within the implementation or used at runtime for detection of relevant attacks (e.g., Swaddler). However, these existing works are limited in two aspects. First, they all require the web application source code for instrumentation, which may not be available in practice. Second, they all infer the application specification at the program level, which makes their approach closely coupled with the programming languages (e.g., PHP, JSP) and frameworks. The correctness and accuracy of their derived specifications are highly dependent on and thus limited by their capability of dealing with language-level details. For example, MiMoSA and Swaddler cannot handle object-oriented programs gracefully.

In this paper, we present BLOCK, a BLack-bOx approach for detecting state violation attaCKs against web applications. Due to the stateless nature of HTTP protocol, session variables are explicitly defined in web applications to maintain the state of a web session. Session variables can be maintained either at the client side (i.e., via cookies, URL rewriting or hidden forms) or at the server side with a session ID issued to the client for indexing. The key idea of BLOCK is to infer the intended behavior model of the web application (i.e., specification) by observing the web request/response sequences and their associated session variable values during attack-free executions. Then, the inferred model is used for evaluating web requests and responses at runtime, combining with current session information. Any web request or response that violates the model is identified as a potential state violation attack and blocked. In particular, we leverage the stateless property of HTTP and regard the vector of current values of session variables as part of the input along with web request to the application, the web responses and the updated session variables as the output. In this way, the web application can be approximated as a stateless system. Under this stateless system model, we characterize the application behavior from three aspects in the form of likely invariants: 1) input invariants, which model the relationship between the web requests and the session variable values, 2) input/output invariants, which capture the relationship between the web request and response as well as the changes in the session variables after the web request is processed, and 3) input/output sequence invariants, which leverage the historical web request/response pair sequences to capture the application states that are not revealed by defined session variables.

To our knowledge, BLOCK is the first black-box technique that addresses state violation attacks towards web applications. Our approach is independent of the application source code and able to handle a variety of programming frameworks. Thus, it can scale up to protect a large number of web applications.

Our contributions are summarized as follows:

- We propose a black-box approach for detecting state violation attacks. We regard the web application as a stateless system and model the relations within web requests, responses and session variables using a set of invariants.

- We implement a prototype of our detection system, which is able to observe and analyze the interactions between the clients and the web application, detect and block state violation attacks.

- We evaluate our detection system using a set of open source web applications. The detection results show that our approach is effective at detecting state violation attacks and incurs acceptable performance overhead.

The rest of this paper is organized as follows. Section II illustrates state violation attacks we target at. Our approach is presented in detail in Section III. The following section describes the implementation of our detection system prototype. Evaluation setup and results are given in Section V. Section VI discusses related works and Section VII concludes this paper.

2. STATE VIOLATION ATTACK

A web application manages the clients' session states to control the access over its restrictive functions and sensitive information, as well as enforce desired state transitions. Although most current web application development frameworks provide session management mechanisms, it is still the developer's responsibility to define and check session variables at appropriate program points, which is usually done in an ad-hoc manner. Three types of vulnerabilities are possibly introduced into the web application: (1) insufficient definition of session variables for differentiating all possible states; (2) insufficient checking of session variables at appropriate program points; (3) erroneous checking of session variables that can be bypassed. They all make the web application vulnerable to state violation attacks (also referred to as the workflow violation attack in Swaddler [7]). The attacker can launch state violation attacks by sending web requests to the web application, which violate the underlying requirements of expected web requests by the developers at the current application state. We use a small PHP web application, as shown in Fig.1, which contains several state management vulnerabilities, as an example to illustrate state violation attacks. This example is also used throughout the paper to demonstrate how we address these attacks.

The first example of state violation attack is authentication/authorization (simplified as auth hereafter) bypass. The web application controls the access over its functions by checking session variables indicating the user privilege before its restrictive functions can be executed. If the application is not at the required state, the web application will redirect the user to the login page, authorization page or an error page. However, if there exists a path leading to the restrictive function with insufficient or erroneous checking of session variables, the attacker is able to bypass the authentication/authorization. The example application demonstrates three cases of auth bypass attacks. *admin.php* and *admin2.php* contain restrictive functions, which should only be accessed by admin users when the session variable *$_SESSION['privilege']* is set to the value of *admin*.

- In *admin.php*, there is no check on the session variable *$_SESSION['privilege']*. The attacker, being either a guest or a regular user, can directly request the page and access the admin functions.

- In *admin2.php*, even though there is an *if* condition check on the session variable $*privilege*, the attacker can append an additional parameter *privilege* to the URL, for example *http://example.com/admin2.php?privilege=admin*, and bypass the auth check. The reason is when the *register_global* option of PHP interpreter is enabled, the parameter attached to the web request will be automatically bound to a global variable, if such variable doesn't exist in the current session state. This vulnerability results from the inappropriate or erroneous check on the session variable.

- In *admin2.php*, even when the auth check fails, the attacker is able to execute the restrictive functions after the redirection (i.e., *header* function) by submitting a POST request with the parameter *title* and change the application's title successfully. This is because there is no *exit* function or an additional check after the redirection.

```
<? php
include_once("header.php");
if (isset($_GET['logout'])){
      session_start();
      unset($_SESSION['username']);
      unset($_SESSION['privilege']);
      session_destroy();
      print "You are logged out.<br>";
} else if (isset($_POST['email'])){
      if (validateLogin($_POST['email'], $_POST['passwd'])){
            $_SESSION['username'] = $_POST['email'];
            if ($_POST['email'] == $admin_email){
                  $_SESSION['privilege'] = "admin";
            } else {
                  $_SESSION['privilege'] = "user";
            }
            header("Location:index.php?username
                        =" . $_SESSION['username']);
            exit();
      } else {
            die("Wrong username or password");
      }
}
?>
<form action='login.php' method=post>
username: <input name="email" type="text"><br>
password: <input name="passwd" type="password"><br>
<input name="submit" type="submit"> </form>
<?php  include_once 'footer.html';?>
```

login.php

```
<?php  include_once 'header.php';
logIdentity();
print "<a href='admin2.php'>Next step: change the
title</a>";
include 'footer.html'; ?>
```

admin.php

```
<? php
include_once 'header.php';
if (isset($_GET['username'])){
      $userid = $_GET['username'];
      showUserInfo($userid);
      if ($_SESSION['privilege'] == "admin"){
            print "<a href='admin.php'>Admin link</a><br >";
      }
}
print "<a href='login.php?logout=1'>Logout</a><br>";
include_once 'footer.html';
?>
```

index.php

```
<? php
include_once 'header.php';
if ($privilege != "admin"){
      header("Location: index.php?username
                  =".$_SESSION['username']);
}
if (isset($_POST['title'])){
      modifyTitle($_POST['title']);
}
?>
<form action='admin2.php' method=post>
New title: <input name="title" type="text"><br>
<input name="submit" type="submit">
</form>
<a href='login.php?logout=1'>Logout</a>
<? php
include_once 'footer.html';?>
```

admin2.php

Figure 1: Example Application

The second example of state violation attack is parameter manipulation. In a lot of cases, the web application assumes implicit relations between the user's input parameters within web requests and the session state. Such a relationship may also be reflected from web responses returned by the web application. If the application doesn't check the session state when accepting the web request, the attacker is able to manipulate the input parameters and gain access to unauthorized information. In the example application, after the user logs in, he/she will be redirected to the *index.php* page, which displays his/her personal information. The web application assumes the request parameter *username* is always equal to the value of session variable *$_SESSION['username']*. If the equality relationship is not examined when the user's personal information is retrieved, the attacker is able to view any user's information by modifying the *username* parameter within the web request.

The third state violation attack is workflow bypass. A web application usually has an intended workflow, which requires the user to perform a predefined sequence of operations to complete a certain task. For example, an e-commerce website has a predefined checkout procedure, which instructs the customer to first fill in the shipping information and then the credit card information before the order can be confirmed and submitted. Such a temporal relationship is enforced by the restrictions over the session state transitions. However, if the session variables are insufficiently defined or checked for guarding the desired state transitions, the attacker is able to bypass certain required steps and violate the intended workflow. The example application requires the admin user first access *admin.php*, which logs his/her identify (by *logIdentity* function) before he/she can modify the application title in *admin2.php*. The two steps indicate two different session states and the transition between them should be guarded by the web application. However, there is no session variable defined for indicating whether the identity of the admin user has been logged or not. The attacker can directly point to *admin2.php* page without his/her identity being logged.

3. APPROACH

Our approach for detecting state violation attacks has two key phases. In the training phase, the intended behavior model of the web application (i.e., the specification) is derived by observing the web request/response sequences and the corresponding session variable values during its attack-free execution. In the detection phase, the inferred model is used to evaluate each incoming web request and outgoing web response and detect any violations.

Due to the stateless nature of HTTP protocol, session variables are explicitly defined in web applications to maintain the state of a web session. There are two ways for maintaining session states: 1) client side only, where session states are directly carried in cookies, hidden forms, or URLs; 2) collaboration of the client and the server, where the server stores the session states and issues a session ID to the client for indexing its session states. In either case, session states can be retrieved at runtime for each web request independent of the web application implementation. For example, when session states are carried in cookies, hidden forms, or rewritten URLs, they can be directly retrieved from the web requests. When session states are kept in the server side, they can be found either in a file or a database table. In the case of PHP, the session state is by default stored in temporary files located at */var/lib/php5*, which is indexed by session ID within web requests, while in the case of JSP, the session state is persisted in database tables.

One straightforward approach to modeling the application behavior is to derive its states from the session variables and their values directly. Yet, this approach has several issues:

1) at one application state, session variables may exhibit a large range of values. For example, *$_SESSION['username']* can assume as many possible values as the number of registered users in the same application state. Thus, directly using session variable values to differentiate application states may result in a large number of spurious states; 2) definition of session variables may be missing from the application implementation. As a result, two application states in the specification can not be differentiated by the collection of all session variables. For example, in the application shown in Fig. 1, there is no session variable defined for indicating whether the admin user identity has been logged.

Our approach follows the stateless property of HTTP and regards the session variables as part of the input to the web application along with web requests. Similarly, the output of the application consists of the web response and the session variables. In this way, the web application can be regarded as a stateless system, as shown in Fig. 2. Under this stateless system model, we characterize the application behavior in the form of three types of likely invariants. 1) Type I input invariants: recall that the web application input consists of the web request and the values of the session variables when the request is made. This type of invariants models the relationship between the web requests and the session variable values. Essentially, it tries to capture the constraints on the web requests at certain session states. By identifying the invariant component of session variables, this approach avoids the introduction of spurious states by unnecessary session variables. 2) Type II input/output invariants: this type of invariants models the relationship between the web request and response as well as the changes in the session variables after the web request is processed. Essentially, it tries to capture the constraints on the application state transition and the input/output dependency at a certain state. Both type I and II invariants rely on the session variables to infer the application states. When the session variables are not sufficiently defined, we need a third type of invariant. 3) Type III input/output sequence invariants: this type of invariants models the relationship between consecutive web request/response pairs. Essentially, it tries to capture the application states that are not revealed by defined session variables by leveraging the historical request/response information. In the following sections, we first formalize our system model and then illustrate how to extract three types of invariants and apply them into runtime detection.

3.1 System Model

As shown in Fig.2, a web application is regarded as a stateless system F, which accepts an input m_{in} and emits an output m_{out}, expressed as $F(m_{in}) = m_{out}$. An input m_{in} consists of a web request and a set of session variable name/value pair $S(m_{in})$. To facilitate detection, we further decompose a web request into two components: a web request key $r(m_{in})$, which includes the HTTP request method and the target file, and a set of input parameter name/value pair $P(m_{in})$. In this paper, we only consider GET and POST methods and focus on PHP pages. For example, the web request keys include *GET-login.php*, *POST-login.php*, in the application shown in Fig. 1. Similarly, an output consists of a web response and a set of session variable name/value pair $S(m_{out})$. A web response is a synthesized web page, which is usually generated by filling dynamic contents into static web page structure (i.e., template). To deal with the infinite number of possible web responses, we decompose a web page into a web template, the number of which is finite, with a set of dynamic contents, which become output parameters. If we assign a unique ID to each static template, a web response can be symbolized as a web template ID (i.e., web response key $v(m_{out})$) and a set of output parameter name/value pair $Q(m_{in})$. In the next section, we illustrate how to symbolize a web page into a web template with a set of output parameters.

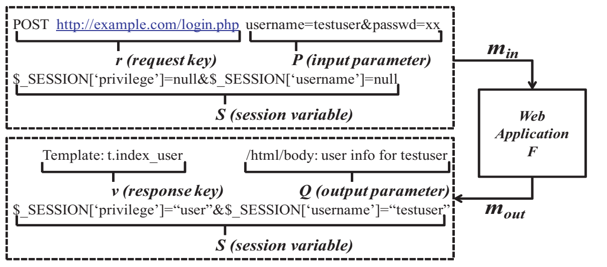

POST http://example.com/login.php username=testuser&passwd=xx
r (request key)
P (input parameter)
$_SESSION['privilege']=null&$_SESSION['username']=null
S (session variable)
m_in
Web Application F
Template: t.index_user
/html/body: user info for testuser
v (response key)
Q (output parameter)
$_SESSION['privilege']="user"&$_SESSION['username']="testuser"
S (session variable)
m_out

Figure 2: A stateless view of web application

3.2 Web Page Symbolization

To symbolize a web page, we first extract the web templates (O) from all the observed web pages (D). Then, given a web page $d \in D$, we classify it into a most possible template (v) and extract the set of output parameters (Q) accordingly. Techniques for extraction of templates from web pages have been presented in existing literatures [19, 13]. In this paper, we leverage the method from TEXT [13], which expresses the DOM tree structure of a web page as a set of essential paths. Our template extraction procedure contains the following four steps. Step 1 and 2 are similar to TEXT and step 3 and 4 are designed to fulfill the purpose of template extraction in our context.

(1) Transformation: the DOM tree structure of a web page d is first transformed into a set of paths P_d. Here, we focus on the paths that lead to the leaf text nodes, which carry the information sent back to the clients within web pages. An index page from our example application can be expressed as three paths: "*/html/body/Welcome to the application*", "*/html/body/user information for: testuser.*" and "*/html/body/a/logout*", as shown in Fig.3.

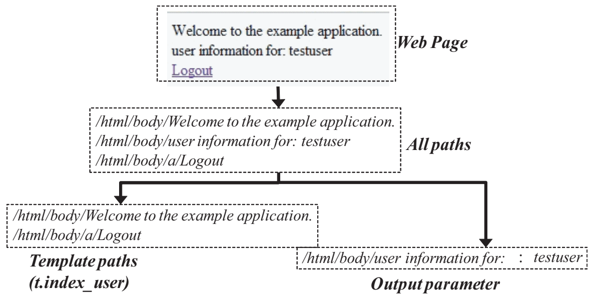

Welcome to the example application.
user information for: testuser
Logout
Web Page
/html/body/Welcome to the example application.
/html/body/user information for: testuser
/html/body/a/Logout
All paths
/html/body/Welcome to the example application.
/html/body/a/Logout
Template paths
(t.index_user)
/html/body/user information for: : testuser
Output parameter

Figure 3: An example of page symbolization

(2) Pruning: to extract templates from all the paths, those paths that lead to dynamic contents should be pruned. To do so, we define the support of a path as the number of pages in D that contain the path. Since the occurrence of a path

that belongs to a template is generally higher, paths with low support are most likely dynamic contents and should be pruned. For each page d, the minimum support threshold t_d is defined as the mode (i.e., the most frequent value) of the occurrence of paths that are contained in the page. Note that using one threshold for all the pages is inappropriate as each template may generate different number of pages. After the paths with support lower than the threshold are pruned, each page is expressed as a set of "essential" paths. We use $ep(d)$ to denote the number of essential paths contained in the page d.

(3) Clustering: two web pages are probably generated from the same web template if they have similar set of essential paths. The similarity ($Dist$) between two pages d_i d_j is defined as follows:

$$Dist(d_i, d_j) = \frac{cp(d_i, d_j)}{\sqrt{ep(d_i) \times ep(d_j)}} \qquad (1)$$

where $cp(d_i, d_j)$ is the number of common essential paths contained in d_i and d_j. We then perform hierarchical agglomerative clustering over all pages based on the above similarity metric. Each resulting cluster corresponds to a web template. The essential path set of a new template is the intersection of path sets from the two templates that are merged together.

(4) Parametrization: for each page in D, after eliminating the essential paths contained in the template it belongs to, the remaining paths in its path set belong to output parameters. The parameter is identified by the path leading to the text node and its value is the content of the text node. We extract those parameters that are observed in all the pages that belong to the template as the set of output parameters of the template. For each parameter, we put the common parts (i.e., tokens) from all the observed values into the parameter name and only extract the variable part as its value.

For the example application, we obtain seven templates. They are the login page (*t.login_form*), logout page (*t.logout*), wrong login page (*t.wrong_login*), regular user information page (*t.index_user*), admin user information page (*t.index_admin*), logging identity page (*t.admin*) and the title change page (*t.title_form*). As shown in Fig.3, the template *t.index_user* has a parameter "*/html/body/user information for:*", which displays the user's information and its value is the current user name.

Given a web page d, it is first transformed into a set of paths. Then, it is classified into the template v that has the highest similarity with its path set (i.e., $v = argmax(Dist(d, v_i))$, $v_i \in O$). The corresponding output parameters for the template are finally extracted.

3.3 Invariant Extraction

We extract three types of invariants: (1) type I input invariants, indexed by the web request key r; (2) type II input/output invariants, indexed by the key pair (r, v); (3) type III input/output sequence invariants, also indexed by the request key r. We also show some example invariants extracted from the application in Fig.1.

3.3.1 Type I Invariant

The inputs with the same request key r are grouped together. We extract the following types of invariants for each request key r.

(1) A set of session variables $S_{inv}(r)$ that are always present. An example invariant of this type is $S_{inv}(GET\text{-}index.php) = \{$*$_SESSION['username']*, *$_SESSION['privilege']*$\}$.

(2) A set of input parameters $P_{inv}(r)$ that are always present. An example invariant of this type is $P_{inv}(POST\text{-}login.php) = \{email, passwd\}$.

(3) For a specific session variable $s \in S_{inv}(r)$, its value is drawn from an enumeration set $V(s, r)$. For example, invariants of this type include: $V($*$_SESSION['privilege']*, *GET-admin.php*$) = \{admin\}$, $V($*$_SESSION['privilege']*, *GET-index.php*$) = \{admin, user\}$;

(4) For a specific input parameter $p \in P_{inv}(r)$, its value is drawn from an enumeration set $V(p, r)$.

(5) The value of an input parameter $p \in P_{inv}(r)$ is always equal to the value of a session variable $s \in S_{inv}(r)$. For the request key *GET-index.php*, the session variable *$_SESSION['username']* is always equal to the input parameter *username*.

3.3.2 Type II Invariant

The input/output pairs with the same key pair (r, v) are grouped together. We first extract the same set of invariants as type I for the key pair. For example, an invariant drawn for the key pair (*GET-login.php*, *t.logout*) is that $V(logout,$ (*GET-login.php*, *t.logout*$) = \{1\}$ and the input parameter *logout* is added into $P_{inv}($*GET-login.php*, *t.logout*$)$.

We also extract two new invariants for each key pair (r, v):

(1) The value of an output parameter is always equal to the value of an input parameter and/or a session variable. This invariant reflects the dataflow within the web application. An invariant for the key pair (*POST-login.php*, *t.index_user*) is that the output parameter */html/body/* of the template *t.index_user* is always equal to the session variable *$_SESSION['username']* and the input parameter *username*.

(2) The session state is unchanged. For example, the user's session state always stays the same by observing the key pair (*GET-login.php*, *t.login_form*), but evolves for the key pair (*POST-login.php*, *t.index_user*).

3.3.3 Type III Invariant

For each request key r, we extract the following invariant:

(1) A set of input/output key pairs that always precede the web request key in one session. An invariant of this type is the key pair (*GET-admin.php*, *t.admin*) always precedes the request key *GET-admin2.php* and the key pair (*GET-admin2.php*, *t.title_form*) always occurs before *POST-admin2.php*.

3.4 Detection

Each web request key r is associated with a set of invariants, including both type I and type III invariants. Each input/output key pair (r, v) is also associated with a set of type II invariants. For detection, each invariant is transformed into an evaluation function, which operates on an input or an input/output pair. If the input or input/output pair satisfies the invariant, the function returns true. Otherwise, the function returns false. The runtime detection is performed in two phases:

(1) validating the input m_{in}: the web request is accepted, if and only if the request key has been observed and all the invariants associated with it are satisfied. Otherwise, the web request is dropped.

(2) validating the input/output pair (m_{in}, m_{out}): the web page is sent back to the user if and only if the corresponding

key pair has been observed and all the invariants associated with it are satisfied. Otherwise, the web page is blocked.

All the attacks that exploit the example application can be detected by our extracted invariants. (1) Each auth bypass attack instance violates the invariants associated with three request keys *GET-admin.php*, *GET-admin2.php* and *POST-admin2.php* respectively and are detected at the first phase. For example, the first attack instance violates the invariant $V(\$\_SESSION[\text{`privilege'}], GET\text{-}admin.php) = \{admin\}$. (2) the parameter manipulation attack violates the invariant associated with the request key *GET-index.php* where the input parameter *username* is always equal to the session variable *$_SESSION['username']* and is detected in the first phase. It also violates the invariant of the key pair (*GET-index.php*, *t.index_user*) that the output parameter "*/html/body/user information for:*" is equal to both the input parameter *username* and *$_SESSION['username']*. (3) the workflow bypass attack violates the invariant associated with the request key *GET-admin2.php* that the key pair (*GET-admin.php*, *t.admin*) always precedes the request key and is detected in the first phase.

4. IMPLEMENTATION

We implement the prototype of our detection system BLOCK as a proxy that sits between the web application and the client, as shown in Fig. 4. BLOCK is capable of intercepting all the messages exchanged between the web application and the client and taking snapshots of the user's session information stored at the server side. To capture the web requests and responses, we build BLOCK on top of WebScarab [18], an open source web application testing tool, which is deployed at the web server and configured as a reverse proxy. PHP web applications, which are our focus in this paper, by default store the users' session information in temporary files at the directory */var/lib/php5*. BLOCK is able to locate the correct session files, indexed by the session ID within the web request, and read the user's session information. BLOCK can be operated in two modes: training and detection. In the training mode, BLOCK collects the observed web requests, responses and their associated session information, analyzes those execution traces and extracts the set of relevant invariants. In the detection mode, BLOCK monitors the interactions between the clients and the web application, dynamically detects and blocks those potential attacks that violate the extracted invariants. We note that BLOCK can be easily extended to other platforms other than PHP by just modifying the component that accesses the session information to handle a variety of programming frameworks. For example, in the case of Tomcat servlet, the component should be able to access database tables via JDBC drivers, which store persistent session information. Our implementation is independent of the web application (i.e., doesn't require the source code for analysis or instrumentation). Thus, it can scale up to protect a large number of web applications.

4.1 Training Mode

The components of BLOCK in the training mode are shown in Fig. 5. Whenever a web request or a web page is captured, the message constructor takes a snapshot of the current session state and composes the corresponding messages, which is sent to the trace collector. After suficient traces have been collected, BLOCK will perform offline learning. The trace processor first extracts web templates from observed web pages, then parses both the input and output messages into the designated format: a request or response key associated with a set of key/value pairs for both parameters and session variables. The parsed traces are fed into the invariant extractor, where all three types of invariants are derived. Especially, the value-related invariants (e.g., the equality relationship between variables, the enumeration value set of variables) are inferred by leveraging Daikon engine [9], a well-known tool for dynamic inference of program invariants. The traces are transformed into the format required by Daikon engine and the output is a set of invariants extracted for each declared entry. Presence-related invariants are extracted by self-developed programs. All extracted invariants comprise the web application's specification.

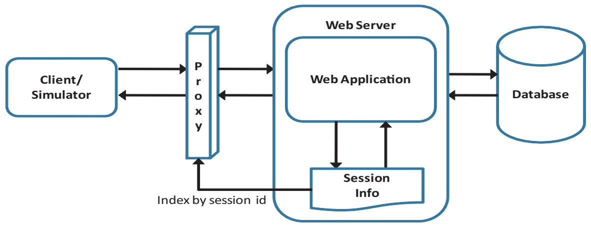


Figure 4: Overview of BLOCK

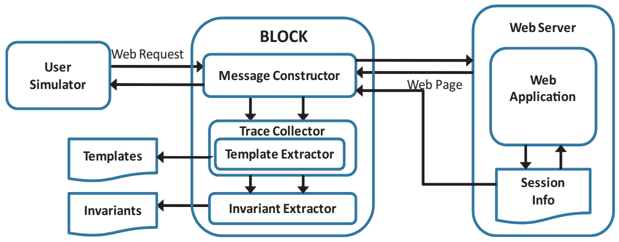


Figure 5: Training Mode

4.2 Detection Mode

Once the invariants are extracted, BLOCK switches to the detection mode, as shown in Fig. 6. The invariant interpreter loads and interprets the extracted invariants. At runtime, the message constructor combines session information with the intercepted web request, composes an input and sends it to the detector for evaluation. If the input is accepted, the web request is forwarded to the web application and logged as the current input for the web application. Otherwise, the web request is dropped. When the message constructor receives a web response, if the response is a redirection, the subsequent web request will not be evaluated or logged. If the response is a web page, the message constructor assigns the web page a response key based on its web template, composes an output and sends it to the detector, where the output is paired with the current input and evaluated. If the output is accepted, the web page is returned to the client and the key pair is logged for the current user session. Otherwise, the web response is blocked and the current input is invalidated. After the user's session has terminated, all of the logged key pairs are cleaned up.

5. EVALUATION

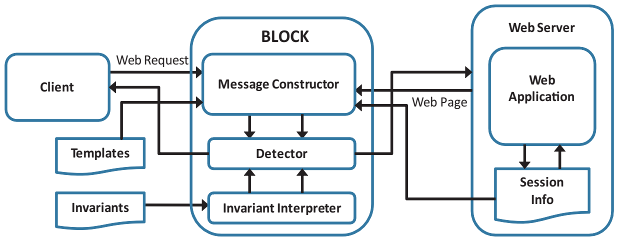


Figure 6: Detection Mode

We evaluate our approach using a set of open source PHP web applications, which are representative with different types of functionalities. (1) Scarf is a conference management system, which is used for managing sessions, papers, users and comments. It is known with an auth bypass vulnerability (CVE-2006-5909). The attacker can directly visit the administrative page *generaloptions.php* and modify the system settings and user accounts, since the admin page doesn't check the privilege of current user. It echoes the first case of auth bypass in the example application. (2) Simplecms is a simple content management system that allows the admin to publish and manage contents. It is also vulnerable to an auth bypass attack in *Auth.php* page (BID 19386). It uses the *register_globals* mechanism insecurely. An attacker can append a parameter *loggedin* to the web request and bypass the authentication check. It echoes the second case of auth bypass in the example application. (3) Bloggit is a blog application that supports web blog management. It also has an auth bypass vulnerability (CVE-2006-7014) in *admin.php* page where the restrictive code continues being executed after the auth check fails. It echoes the third case of auth bypass in the example application. (4) Wackopicko [24] is an online photo sharing website that allows users to upload pictures, comment on and purchase other people's pictures, etc. It is initially written for testing web application vulnerability scanners. It is designed with a number of vulnerabilities, such as cross-site scripting, SQL injection, file inclusion, etc. Here, we focus on its parameter manipulation vulnerability. After a user logs in, he/she can view the personal information in *home.php* page. However, an attacker can manipulate the *userid* parameter to view any other user's information and owned pictures. (5) OsCommerce [17] is a widely-used open source e-commerce application. To evaluate our approach of handling workflow bypass attacks, we instrument one vulnerability into the checkout procedure, which allows the attacker to directly go to the payment page without selecting the shipping method and the total charge doesn't include shipping fees. Table 1 shows a summary of web applications we use for evaluation.

All the web applications and BLOCK (based on WebScarab) are deployed on a 2.13GHz Core 2 Linux server with 2GB RAM, running Ubuntu 10.10, Apache web server (version 2.2.16) and PHP (version 5.3.3). To collect training traces, each web application is driven by a user simulator, which emulates the interactions between a normal user and the web application. For each web application, user roles and atomic operations are first identified manually. Then, the user simulator is developed based on the Selenium webdriver [21] to emulate a normal user operating a web application. The simulator leverages a library of user information

Table 1: Summary of Evaluated Web Applications

Application	PHP files	Description	Vulnerability
Scarf	21	Conference management system	Auth bypass (CVE-2006-5909)
Simplecms	23	Content management system	Auth bypass (BID 19386)
BloggIt	24	Blog engine	Auth bypass (CVE-2006-7104)
Wackopicko	53	Photo sharing website	Parameter manipulation
OsCommerce	533	Open source e-commerce solution	Workflow bypass (instrumented)

of all the undergraduate students from a network security class and is able to automatically explore the web application, such as clicking the links, filling in and submitting forms. Among the available atomic operations for the currently chosen user, it randomly selects one as the emulated user's next step. The user simulator is set up at a 2.83GHz Core 2 desktop with 8GB RAM running Windows 7 and Firefox 4. The client is connected to the web server using Ethernet.

5.1 Detection Effectiveness

BLOCK first runs in the training mode to collect the execution traces, generated by the user simulators. Table 2 shows the summary of our collected traces [1]. Then, it analyzes those traces, extracts web request keys, web page templates, as well as all three types of invariants. To observe the impact of the training set size on the number of derived invariants, we vary the training set size and calculate the resulting invariants. Fig. 7 shows the experiment result we obtain for the Scarf application. We can see that the numbers of type I and III invariants initially decrease and then converge with the increase of training set size, indicating the elimination of false invariants learnt from insufficient training samples. The number of type II invariants first increases, due to the exploration of new state space that has not been revealed by the small training set, then also slowly converges. Based on this observation, we use the training set for each application where the number of invariants converges.

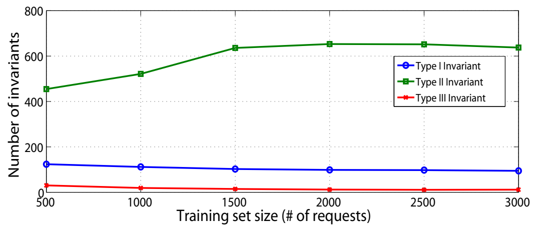


Figure 7: Number of invariants vs. Training set size (Scarf application)

[1]Here, we note that our training only covers the part of most used functions for customers in OsCommerce application. Also, we don't count redirection headers as web pages.

Table 2: Summary of Training Set

Application	Requests	Web Pages	Request Keys	Web Templates	Key Pairs	Type I Inv	Type II Inv	Type III Inv
Scarf	3225	3200	21	26	69	90	640	11
Simplecms	2661	2555	17	12	34	56	190	28
BloggIt	2657	2645	16	13	47	65	377	9
Wackopicko	2949	2946	20	12	30	36	155	37
OsCommerce	3879	3444	25	36	123	374	4609	26

Then BLOCK switches to the detection mode. The clean test set is generated by both the user simulators and the undergraduate students who manually operate the web applications. Ten attack instances are manually generated under different circumstances against each web application. Table 3 shows the summary of the test set and all the detection results. All of the attacks are successfully detected by BLOCK and the false positives for both web requests and responses are fairly low. This fact demonstrates the effectiveness of our approach at detecting state violation attacks.

We further investigate those false positives and find out two major sources. One is the incomplete exploration of the web application performed by the user simulator. The capability of the user simulator determines the state space that our detection system can characterize for the web application. The more the simulator explores, the richer and more accurate these invariants are. In our evaluation, some false positives result from error pages that are not explored by the simulator, thus not observed and profiled by the invariant extractor. In practice, if real-world traces are available, our detection system can be readily applied and work effectively. The other source of false positives is the inaccurate symbolization of web pages. Page symbolization affects both the training and detection phase. In the training phase, both the number and the quality of the inferred invariants, especially for type II, are closely related with the number of extracted web templates. We can see that the number of type I and III invariants converges very fast, thus leading to an extremely low number of false positives for web requests, while type II invariants bring more false positives of web responses. In the detection phase, due to the content drift of web pages, it is possible that a web page is classified into a wrong template, which likely results in an unobserved pair of input/output and thus a false positive. We use the same clustering threshold for all applications to extract web templates, which also introduces certain level of inaccuracies. Since web template extraction is not our focus in this paper, we adapt the methods from TEXT [13] and it works well with the web application we use for evaluation. To increase the accuracy and robustness of web page symbolization, advanced algorithms or manual audit can be introduced for guiding the process.

The detection results also show the types of invariants violated by different attacks. Auth bypass attacks on insufficient checking of session variables result in violations of type I invariants that are imposed on the session state, when web requests are received. They would also violate type III invariants due to the missing step of authentication/authorization. Parameter manipulation attacks can be detected by type I invariants, if the input parameters are related to the session variables. They may also be identified by type II invariants, if the corresponding web pages contain output parameters that are related with the session state. Workflow bypass attacks will be blocked in the same manner as auth bypass attacks, if the session variables, which are used for guarding the state transitions, are not checked. If there are no such guarding session variables, e.g., in the example application, type III invariants would help to identify workflow bypass attacks due to the constraints imposed on the sequence of operations.

5.2 Performance Overhead

Since our detection system sits between the client and the web application, it will affect the response time of the web application. First, the WebScarab proxy intercepts and forwards all the messages exchanged between the user and the web application, which would increase the response time. Second, the integrated detector evaluates the web requests and web pages, which also introduces additional delay. To measure the performance overhead brought by our detection system, we use the simulators to perform a designated sequence of operations and log the response time for every web request. We compare the performance under three configurations: 1) without WebScarab proxy, 2) with WebScarab proxy deployed but the detector disabled, 3) with WebScarab proxy deployed and the detector enabled. Figure 8 shows the summary of the averaged response time for each application under the above three scenarios. We can see that the average response time increases by a factor from 1.5 to around 5, if BLOCK is deployed and enabled. While the resulting response time is still acceptable, we notice that more than 90% of the overhead is brought by the WebScarab proxy and only a small amount is introduced by the detector. For our current prototype implementation, no modifications or configurations are made to the WebScarab proxy to enhance its performance. If a more light-weight and efficient proxy (e.g, Apache mod_proxy) is employed for integrating our detection system, it is possible to reduce the response time, which serves as our future work.

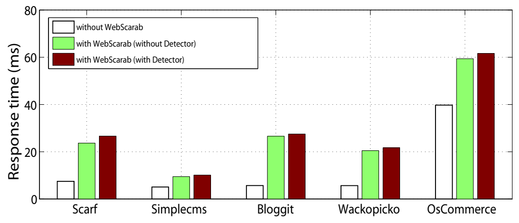


Figure 8: Summary of performance overhead

Table 3: Summary of Detection Result

Application	Requests (clean test set)	Web pages (clean test set)	Blocked requests (false positive)	Blocked responses (false positive)	Attacks	Detected	Invariant violations
Scarf	1364	1360	0	6	10	10	type I and III
Simplecms	1731	1688	0	8	10	10	type I and III
BloggIt	1044	1024	0	0	10	10	type I and III
Wackopicko	1322	1314	0	1	10	10	type I and II
OsCommerce	1505	1460	3	10	10	10	type I and III

5.3 Discussion

There is one limitation of BLOCK we would like to point out. BLOCK only observes and models the relations between web requests, web responses and the session variables. Thus it cannot handle the attacks that violate the persistent states that exist in database tables. If BLOCK is extended to capture and analyze the SQL queries/responses from a database, it has the potential to handle this type of state violation attack. This serves as our future work.

Our technique bears the same limitations as other dynamic analysis techniques. The completeness and correctness of inferred invariants cannot be guaranteed. In order to put BLOCK into practice, introducing some manual intervention is preferable to guarantee sufficient training and suppress false positives. In the future, we would like to investigate mechanisms for automatic verification of likely invariants.

6. RELATED WORK

Our work falls within the category of web application security and our approach is closely related to the specification inference of software.

6.1 Web Application Security

Web application security has been a popular research topic these years. A large body of existing works investigate input validation attacks, such as cross-site scripting, SQL injection, which exploit the applications' insufficient or erroneous sanitization of the user inputs. Compared with state violation flaws, which is the focus of this paper, input validation flaws are independent of the application logic and thus can be captured via a general specification. For example, the information flow model has been applied to the input validation problem, where a set of data input points are defined as sources, and the security sensitive operations are modeled as sinks. Based on this model, both static and dynamic program analysis techniques are employed to identify the insufficient or erroneous sanitizations within the web application, which result in insecure information flow [12]. It is worth noting that the black-box approach [20], techniques that analyze the external request/response flow [1, 23], and approaches of inferring a DFA for web requests [15, 11] have been presented to address input validation attacks. Black-box techniques have also been applied to address other problems within web applications, such as post-migration testing (Splitter [8]), insider threats (CADS [6]), form tampering (NoTamper [5]) and HTTP parameter pollution [3]. However, due to different nature of problems, they don't take into account the internal state of the web application and can not be applied to state violation attacks.

MiMoSA [4] and Waler [10] employ white-box analysis techniques to identify vulnerabilities within web applications that attract state violation attacks. While they may achieve better accuracy (i.e., less false positives) than black-box techniques, their capability is limited in that they rely on precise modeling of the application source code and programming frameworks, which is difficult and not scalable. The most related work to ours is Swaddler [7], which also detects state violation attacks at runtime by evaluating the deviations of session variables when entering a specific program block. One major deficiency of Swaddler is that it completely depends on user defined session variables. In cases where insufficient session variables are defined, as shown in the example application, it cannot detect those attacks. In contrast, our Type III invariants that are defined based on the web request/response history can capture the application state that is not revealed by defined session variables. Thus even when the session variables are insufficient or unreliable, our approach is still effective.

6.2 Specification Inference of Software

Software specification is essential for verification of program behaviors and program testing. However, a complete and machine understandable specification is rarely available. Thus, researchers are motivated to study the problem of inferring software specifications. Static inference techniques analyze the program code to extract the partial orders of function calls [14], while dynamic inference techniques try to profile the program behavior through mining program execution traces. Daikon engine [9], the most famous tool in this field, extracts value-related invariants by matching invariant templates to expressions. Strauss [2] formalizes the specification mining as a grammar inference problem and learns probabilistic finite state automata (PFSA) from traces. Perracotta [26] mines two-letter alternating patterns of functions from imperfect traces. Gk-tail [16] builds extended finite state machine (EFSM) combining both value-related and temporal properties. Our approach falls into the category of dynamic inference techniques. Different from these generic software specification inference methods, our work leverages the unique stateless feature of HTTP protocol and its associated session management mechanism and can be applied to distributed client/server web applications.

7. CONCLUSION

This paper presents BLOCK, a black-box approach for detecting state violation attacks, and evaluates its prototype implementation using a set of open source PHP web applications. The results validate the effectiveness of BLOCK. Our approach is valuable in that it is independent of the web application source code and can fit into a large variety of web application hosting scenarios based on different application frameworks, where the source code may not be available.

Acknowledgment

This work was supported by NSF TRUST (The Team for Research in Ubiquitous Secure Technology) Science and Technology Center (CCF-0424422). We would like to thank Ryan Burns, Brandon Conway and Russ Amos for their help in developing user simulators and Vanderbilt ITS for valuable discussion.

8. REFERENCES

[1] M. Almgren, H. Debar, and M. Dacier. A lightweight tool for detecting web server attacks. In *Proceedings of the ISOC Symposium on Network and Distributed Systems Security*, pages 157–170, 2000.

[2] G. Ammons, R. Bodlík, and J. R. Larus. Mining specifications. In *Symposium on Principles of Programming Languages*, volume 37, pages 4–16, 2002.

[3] M. Balduzzi, C. Gimenez, D. Balzarotti, and E. Kirda. Automated discovery of parameter pollution vulnerabilities in web applications. In *NDSS'11: Proceedings of the 18th Network and Distributed System Security Symposium*, 2011.

[4] D. Balzarotti, M. Cova, V. V. Felmetsger, and G. Vigna. Multi-module vulnerability analysis of web-based applications. In *CCS'07: Proceedings of the 14th ACM conference on Computer and communications security*, pages 25–35, 2007.

[5] P. Bisht, T. Hinrichs, N. Skrupsky, R. Bobrowicz, and V. N. Venkatakrishnan. NoTamper: automatic blackbox detection of parameter tampering opportunities in web applications. In *CCS'10: Proceedings of the 17th ACM conference on Computer and communications security*, pages 607–618, 2010.

[6] Y. Chen and B. Malin. Detection of anomalous insiders in collaborative environments via relational analysis of access logs. In *CODASPY '11: Proceedings of the first ACM conference on Data and application security and privacy*, pages 63–74, 2011.

[7] M. Cova, D. Balzarotti, V. Felmetsger, and G. Vigna. Swaddler: An Approach for the Anomaly-based Detection of State Violations in Web Applications. In *RAID'07: Proceedings of the 10th International Symposium on Recent Advances in Intrusion Detection*, pages 63–86, 2007.

[8] X. Ding, H. Huang, Y. Ruan, A. Shaikh, B. Peterson, and X. Zhang. Splitter: a proxy-based approach for post-migration testing of web applications. In *EuroSys'10: Proceedings of the 5th European conference on Computer systems*, pages 97–110, 2010.

[9] M. D. Ernst, J. Cockrell, W. G. Griswold, and D. Notkin. Dynamically discovering likely program invariants to support program evolution. *IEEE Transactions on Software Engineering*, 27(2):99–123, Feb. 2001.

[10] V. Felmetsger, L. Cavedon, C. Kruegel, and G. Vigna. Toward Automated Detection of Logic Vulnerabilities in Web Applications. In *USENIX'10: Proceedings of the 19th conference on USENIX Security Symposium*, pages 143–160, 2010.

[11] K. L. Ingham, A. Somayaji, J. Burge, and S. Forrest. Learning dfa representations of http for protecting web applications. *Computer Networks and Isdn Systems*, 51:1239–1255, 2007.

[12] N. Jovanovic, C. Kruegel, and E. Kirda. Pixy: A static analysis tool for detecting web application vulnerabilities (short paper). In *S&P'06: Proceedings of the 27th IEEE Symposium on Security & Privacy*, pages 258–263, 2006.

[13] C. Kim and K. Shim. Text: Automatic template extraction from heterogeneous web pages. *IEEE Trans. Knowl. Data Eng.*, 23(4):612–626, 2011.

[14] T. Kremenek, P. Twohey, G. Back, A. Ng, and D. Engler. From uncertainty to belief: inferring the specification within. In *OSDI '06: Proceedings of the 7th symposium on Operating systems design and implementation*, pages 161–176, 2006.

[15] C. Kruegel and G. Vigna. Anomaly detection of web-based attacks. In *CCS'03: Proceedings of the 10th ACM conference on Computer and communications security*, pages 251–261, 2003.

[16] D. Lorenzoli, L. Mariani, and M. Pezzè. Automatic generation of software behavioral models. In *ICSE '08: Proceedings of the 30th international conference on Software engineering*, pages 501–510, 2008.

[17] OsCommerce Inc. http://www.oscommerce.com/.

[18] OWASP WebScarab Project. https://www.owasp.org/index.php/category:owasp_webscarab_project.

[19] D. C. Reis, P. B. Golgher, A. S. Silva, and A. F. Laender. Automatic web news extraction using tree edit distance. In *WWW '04: Proceedings of the 13th international conference on World Wide Web*, pages 502–511, 2004.

[20] R. Sekar. An efficient black-box technique for defeating web application attacks. In *NDSS'09: 16th Annual Network and Distributed System Security Symposium*, 2009.

[21] SeleniumHQ: Web Application Testing System. http://seleniumhq.org/.

[22] Symantec internet security threat report 2009. http://www.symantec.com/business/threatreport/.

[23] G. Vigna, W. Robertson, V. Kher, and R. A. Kemmerer. A stateful intrusion detection system for world-wide web servers. In *ACSAC'03: Proceedings of the Annual Computer Security Applications Conference*, pages 34–43, 2003.

[24] Wackopicko. https://github.com/adamdoupe/wackopicko.

[25] G. Wassermann and Z. Su. Static detection of cross-site scripting vulnerabilities. In *ICSE'08: ACM/IEEE 30th International Conference on Software Engineering*, pages 171–180, 2008.

[26] J. Yang, D. Evans, D. Bhardwaj, T. Bhat, and M. Das. Perracotta: mining temporal api rules from imperfect traces. In *ICSE '06: Proceedings of the 28th international conference on Software engineering*, pages 282–291, 2006.

A Server- and Browser-Transparent CSRF Defense for Web 2.0 Applications*

Riccardo Pelizzi and R. Sekar
Stony Brook University

ABSTRACT

Cross-Site Request Forgery (CSRF) vulnerabilities constitute one of the most serious web application vulnerabilities, ranking fourth in the CWE/SANS Top 25 Most Dangerous Software Errors. By exploiting this vulnerability, an attacker can submit requests to a web application using a victim user's credentials. A successful attack can lead to compromised accounts, stolen bank funds or information leaks. This paper presents a new server-side defense against CSRF attacks. Our solution, called jCSRF, operates as a server-side proxy, and does not require any server or browser modifications. Thus, it can be deployed by a site administrator without requiring access to web application source code, or the need to understand it. Moreover, protection is achieved without requiring web-site users to make use of a specific browser or a browser plug-in. Unlike previous server-side solutions, jCSRF addresses two key aspects of Web 2.0: extensive use of client-side scripts that can create requests to URLs that do not appear in the HTML page returned to the client; and services provided by two or more collaborating web sites that need to make cross-domain requests.

1. INTRODUCTION

The stateless nature of HTTP necessitates mechanisms for maintaining authentication credentials across multiple HTTP requests. Most web applications rely on cookies for this purpose: on a successful login, a web application sets a cookie that serves as the authentication credential for future requests from the user's browser. As long as this login session is active, the user is no longer required to authenticate herself; instead, the user's browser automatically sends this cookie (and all other cookies set by the same server) with every request to the same web server.

The same origin policy (SOP), enforced by browsers, ensures confidentiality of cookies: in particular, it prevents one web site (say, `evil.com`) from reading or writing cookies for another site (say, `bank.com`). However, browsers enforce no restrictions on outgoing requests: if a user visits an `evil.com` web page, possibly because of an enticing email (see Figure 1), a script on this page can send a request to `bank.com`. Moreover, the user's browser will automatically include `bank.com`'s cookies with this request, thus enabling Cross-site Request Forgery (CSRF). A CSRF attack thus enables one site to "forge" a user's request to another site. Using this attack, `evil.com` may be able to transfer money from the user's account to the attacker's account [24]. Alternatively, `evil.com` may be able to reconfigure a firewall protecting a home or local area network, allowing it to connect to vulnerable services on this network [2, 3].

Since CSRF attacks involve cross-domain requests, a web application can thwart them by ensuring that every sensitive request originates from its own pages. One easy way to do this is to rely on the `Referrer` header of an incoming HTTP request. This information is supplied by a browser and cannot be changed by scripts, and can thus provide a basis for verifying the domain of the page that originated a request. Unfortunately, referrer header is often suppressed by browsers, client-side proxies or network equipment due to privacy concerns [1]. An alternative to the `Referrer` header, called `Origin` header [1], has been proposed to mitigate these privacy concerns, but this header is not supported by most browsers. As a result, it becomes the responsibility of a web application developer to implement mechanisms to verify the originating web page of a request.

A common technique for identifying a same-origin request is to associate a nonce with each web page, and ensuring that all requests from this page will supply this nonce as one of the parameters. Since the SOP prevents attackers from reading the content of pages from other domains, they are unable to obtain the nonce value and include it in a request, thus providing a way to filter them out. Many web application frameworks further simplify the incorporation of this technique [23, 8, 12]. Nevertheless, it is ultimately the responsibility of a web application developer to incorporate these mechanisms. Unfortunately, some web application developers are not aware of CSRF threats and may not use these CSRF prevention techniques. Even when the developer is aware of CSRF, such a manual process is prone to programmer errors — a programmer may forget to include the checks for one of the pages, or may omit it because of a mistaken belief that a particular request is not vulnerable. As a result, CSRF vulnerabilities are one of the most commonly reported web application vulnerabilities, and is listed as the fourth most important software vulnerability in the

*This work was supported in part by ONR grant N000140710928, NSF grant CNS-0831298, and AFOSR grant FA9550-09-1-0539.

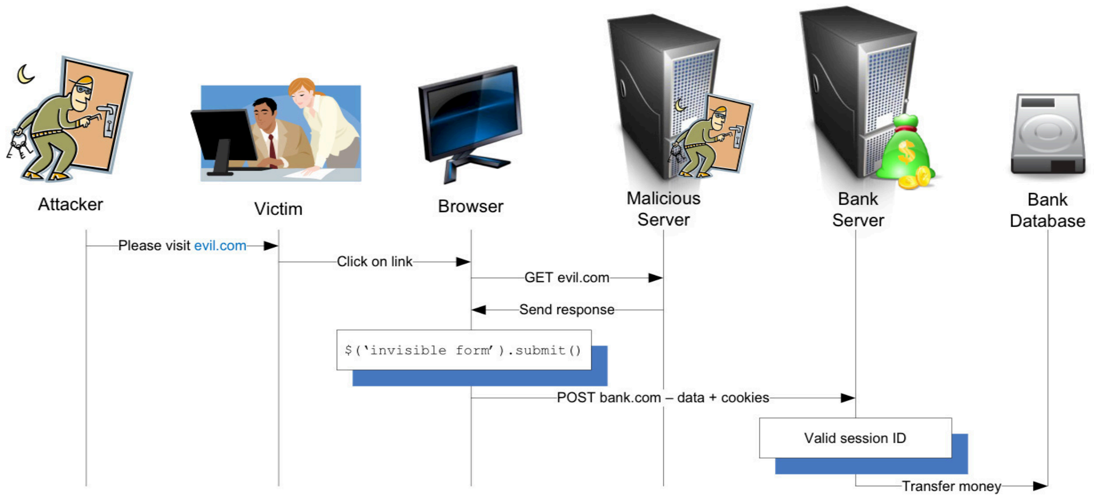


Figure 1: Illustration of a CSRF attack

CWE Top 25 list [6].

Prompted by the prevalence of CSRF vulnerabilities and their potential impact, researchers have developed techniques to retrofit CSRF protection into existing applications. NoForge [16] implements CSRF protection using the same basic nonce-based approach outlined earlier. On the server-side, it intercepts every page sent to a client, and rewrites URLs found on the page (including hyperlinks and form destinations) so that they supply the nonce when requested. RequestRodeo [15] is conceptually similar but is deployed on the client-side rather than the server side. Unfortunately, since these techniques rely on static rewriting of link names, they don't work well with Web 2.0 applications that construct web pages dynamically on the browser. More generally, existing CSRF defenses suffer from one or more of the following drawbacks:

1. *Need for programmer effort and/or server-side modifications.* Many existing defenses are designed to be used by programmers during the software development phase. In addition to requiring programmer effort, they are often specific to a development language or server environment. More importantly, they cannot be deployed by a site administrator or operator that doesn't have access to application source code, or the resources to undertake code modifications.
2. *Incompatibility with existing browsers.* Some techniques require browser modifications to provide additional information (e.g., the referrer or origin [1] header), while others rely on browsers enforcement of policies on cross-origin requests (e.g., NoScript [18], CsFire [7], SOMA [19], RequestRodeo [15]). These approaches thus leave server administrators at the mercy of browser vendors and users, who may or may not be willing to adopt these browser modifications.
3. *Inability to protect dynamically generated requests.* Existing server-side defenses, including NoForge [16], CSRFMagic [23], and CSRFGuard [22], do not work with requests that are dynamically created as a result of JavaScript execution on a browser.
4. *Lack of support for legitimate cross-origin requests.* Previous server-side token-based schemes similar to NoForge are aimed at identifying same-origin requests. However, there are many instances where one domain may trust another, and want to permit cross-origin requests from that domain. Such cross-origin requests are not supported by existing server-side solutions, and there does not seem to be any natural way to extend them to achieve this.

We therefore present a new approach for CSRF defense that does not suffer from any of the above drawbacks. Our solution, called jCSRF, is implemented in the form of a server-side proxy. Note that on web servers such as Apache that support a plug-in architecture, jCSRF can be implemented as a web server module, thus avoiding the drawbacks associated with proxies such as additional performance overheads and HTTPS compatibility.

jCSRF operates by interposing transparently on the communication between clients and servers, modifying them as needed to protect against CSRF attacks. As a server-side proxy, it avoids any need for server-side changes. jCSRF also avoids client-side changes by implementing client-side processing using a script that it injects into outgoing pages. It can protect requests for resources that are already present in the web page served to a client, as well as requests that are dynamically constructed subsequently within the browser by scripts. Finally, it incorporates a new protocol that enables support of legitimate cross-domain requests.

jCSRF protects all POST requests automatically, without any programmer effort, but as we describe later, it is difficult (for our technique and those of others) to protect against GET-based CSRF without some programmer effort. Moreover, GET-requests are supposed to be free of side-effects as per RFC2616 [9], in which case they won't be vulnerable to CSRF. For these reasons, jCSRF currently does not protect against GET-based CSRF.

2. APPROACH OVERVIEW

As described before, the essence of CSRF is a request to a web server that originates from an unauthorized page. We use the terms *target server,* and *protected server* to refer to such a server that is targeted for a CSRF attack. An authorized page is one that is from the same web server ("same-origin request"), or from a second server that is deemed acceptable by this server ("authorized cross-origin request"). In the former case, no special configuration of jCSRF is needed, but in the latter case, we envision the use of a configurable whitelist of authorized sources for a cross-origin request.

We have implemented jCSRF as a server-side proxy, but it can also be implemented as a server-side module for webservers that support modules, such as Apache. This proxy is transparent to web applications as well as clients (webbrowsers), and implements a server- and browser-independent method to check if the origin of a request is authorized. Conceptually, this authorization check involves three steps:

- In the first step, an authentication token is issued to pages served by the protected server.
- In the second step, a request is submitted to jCSRF, together with the authentication token.
- In the third step, jCSRF uses the authentication token to verify that the page from which the request originated is an authorized page. If so, the request is forwarded to the web server. Otherwise, the request is forwarded to the server *after stripping all the cookies*.

Note that stripping off all cookies will cause an authentication failure within the web application, except for requests requiring no authentication, e.g., access to the login page of the web application, or another informational page that contains no user-specific information. Thus, jCSRF is secure by design and will prevent CSRF attacks. Specifically, its security relies only on three factors: unforgeability of authentication tokens, secure binding between the token and the original page, and the correctness of the authorization policy used in the third step. Other design or implementation errors may lead to false positives (i.e., legitimate requests being denied) but not false negatives.

Note that conceptually, the first two steps are similar to those used in previous defenses such as NoForge [16]. Thus, the key novelty in our approach is the design of protocols and mechanisms that ensure that CSRF protection can be achieved for:

- *dynamically created requests:* requests that are constructed as a result of script execution on the client (web-browser). Such requests are common in Web 2.0 applications using AJAX.
- *cross-origin requests:* requests from a web page served by one web site A to another website B, provided B trusts A for this purpose.

When requests are dynamically created, the strategy used by NoForge of statically rewriting the links (to include an authentication token) is not applicable. We have therefore developed a new approach that uses injected JavaScript to carry out this function. In particular, when a page is served by a web application, jCSRF injects some JavaScript code, called jCSRF-script, into this page. On the browser, jCSRF-script is responsible for obtaining the authentication token, and supplying it together with every request originating from this page. By comparing the domain of the current page and the domain of a request, this script can distinguish between same-origin and cross-origin requests, and use different means to obtain the authentication tokens in each case.

We also point out that a static rewriting strategy does not provide a way to validate cross-origin requests. In particular, if a server A embeds a cross-origin request for server B in its page, then the client would need a token for accessing B, but the server A has no easy way to obtain such a token. Note that it cannot directly request such a token from B since the token would have to be bound in some way to the *user's cookies* for B, and A has no access to these cookies. In contrast, we develop a protocol that can support cross-origin requests naturally.

From a conceptual point of view, jCSRF approach can be applied to both GET and POST requests. However, in practice, the "authorized origin" constraint, which forms the basis of all CSRF defense mechanisms, should not be imposed on many GET-requests. Examples include (a) login pages and other pages that contain no security-sensitive data, and (b) book-marked pages, which may or may not contain sensitive data. Application-specific configuration would be required to list such *landing pages* (case (a)) for each application, and except them from authorization checks. Handling case (b) would require some level of browser cooperation, something we do not assume in our work. Moreover, since it is recommended practice to avoid side-effects in GET-request (as per RFC2616 [9]), they are less likely to be vulnerable as compared to POST-requests. Finally, certain HTML elements such as `img` and `frame` cause the browser to issue a GET request before jCSRF-script has a chance to add the authentication token, requiring jCSRF-script to resubmit the requests for these elements and complicate its logic. For these reasons, in our current implementation of jCSRF, GET requests are not subjected to the "authorized origin" constraint.

Below, we provide more details on the key steps in jCSRF.

2.1 Injecting jCSRF-script into web pages

When a page is served by a protected server, jCSRF-proxy automatically injects jCSRF-script into the page. This can be done without having to perform the complex task of parsing full HTML. Instead, the new script is added by inserting a line of the form

```
<script type="text/javascript" src=...  ></script>
```

immediately after the `<head>` tag. Also, jCSRF-proxy includes a new cookie in the HTTP response (unless one exists already) that can be used by jCSRF-script to authenticate same-origin requests. The rest of the page is neither examined nor modified by jCSRF-proxy. As a result, the proxy does not know whether the page contains any cross-origin (or same-origin) requests. It is left to the jCSRF-script to determine on the client-side whether a request being submitted is a same-origin or cross-origin request.

If jCSRF-proxy is implemented as a stand-alone proxy, then it may not be easy to handle HTTPS requests as the proxy will now intercept encrypted content. Although this can potentially be rectified by terminating the SSL sessions at the proxy, a simpler and more preferable alternative is to implement the proxy's logic as a module within the web server.

2.2 Protocol for Validating Requests

Although there is just a single protocol that uses different mechanisms to validate cross-origin and same-origin requests, it is easier to describe them separately. We first

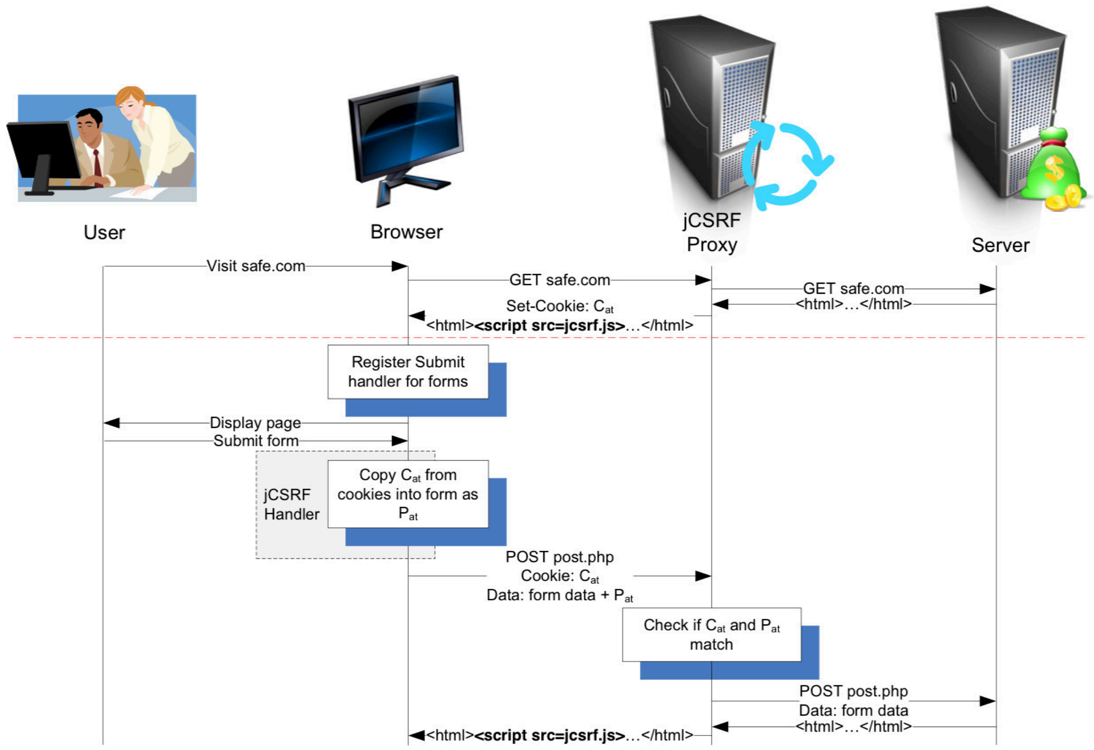

User
Browser
jCSRF Proxy
Server
Visit safe.com
GET safe.com
GET safe.com
<html>…</html>
Set-Cookie: C_{at}
<html><script src=jcsrf.js>…</html>
Register Submit handler for forms
Display page
Submit form
jCSRF Handler
Copy C_{at} from cookies into form as P_{at}
POST post.php
Cookie: C_{at}
Data: form data + P_{at}
Check if C_{at} and P_{at} match
POST post.php
Data: form data
<html>…</html>
<html><script src=jcsrf.js>…</html>

Figure 2: Same-Origin Protocol Workflow

describe the same-origin validation since it is easier to understand, and then proceed to describe the cross-origin case.

2.2.1 *Same-Origin Protocol.*

The same-origin protocol, illustrated in Figure 2, is a simple stateless protocol which authenticates same-origin requests. Red dotted lines in the figure demarcate request-response cycles.

Initially, an authentication token needs to be issued to authorized pages. Since jCSRF permits POST requests only from authorized pages, the very first request from a user has to be a GET request. Such a request is characterized by the fact that a cookie C_{at} used by jCSRF is not set. The server's response to this request is modified by jCSRF-proxy to set this cookie to a cryptographically secure random value. In addition, jCSRF-proxy also injects jCSRF-script in the response as previously described. When this page is received by the browser, jCSRF-script executes, and will ensure (as described further in Section 2.3) that the value of C_{at} is copied into a new parameter P_{at} for all requests originating from this page.

Note that all pages returned by a protected server are modified as above, not just the initial GET request. As such, subsequent requests can provide C_{at} as well as P_{at}. This information is then used in the second step of the protocol in Figure 2 to validate POST requests. In particular, jCSRF-proxy checks if $P_{at} = C_{at}$, and if so, the request is forwarded to the server after stripping out P_{at}. A missing P_{at} or C_{at}, or if $P_{at} \neq C_{at}$, it is deemed an unauthorized request. In this case, jCSRF-proxy strips off all cookies before the request is forwarded to the server. Since web applications typically use cookies to store authentication data, this ensures that the request will be accepted only if it requires no authentication. Note that cross-origin GET requests can be limited in the same way as POST requests, but for reasons described before, the current implementation of jCSRF does not do so.

Correctness.

In order to protect against CSRF, this protocol needs to guarantee the following properties:

- scripts running on an attacker-controlled page visited by user's browser cannot obtain the authentication token for the protected domain.
- any token that may be obtained by the attacker, say, using his own browser, cannot be used to authenticate a request from user's browser to the protected domain
- the attacker should not be able to guess an authentication token that is valid for the protected domain

The first property is immediate from the SOP: since the authentication token is stored as a cookie, attacker's code running on the user's browser runs on a different domain and has no access to it.

The second property holds because the attacker, apart from being prevented by the SOP from reading the token, is also prevented from setting the token. Therefore, any token obtained by the attacker and embedded into forms sent by the user would not match the cookie that jCSRF-proxy previously set for the user.

The third property is ensured by the fact that the authentication token is randomly chosen from a reasonably large keyspace. Specifically, jCSRF-proxy for a server S generates C_{at} as follows. First, a 128-bit random value IR is generated from a true random source, such as `/dev/random`. A pseudo-random number generator, seeded with IR, is then used to

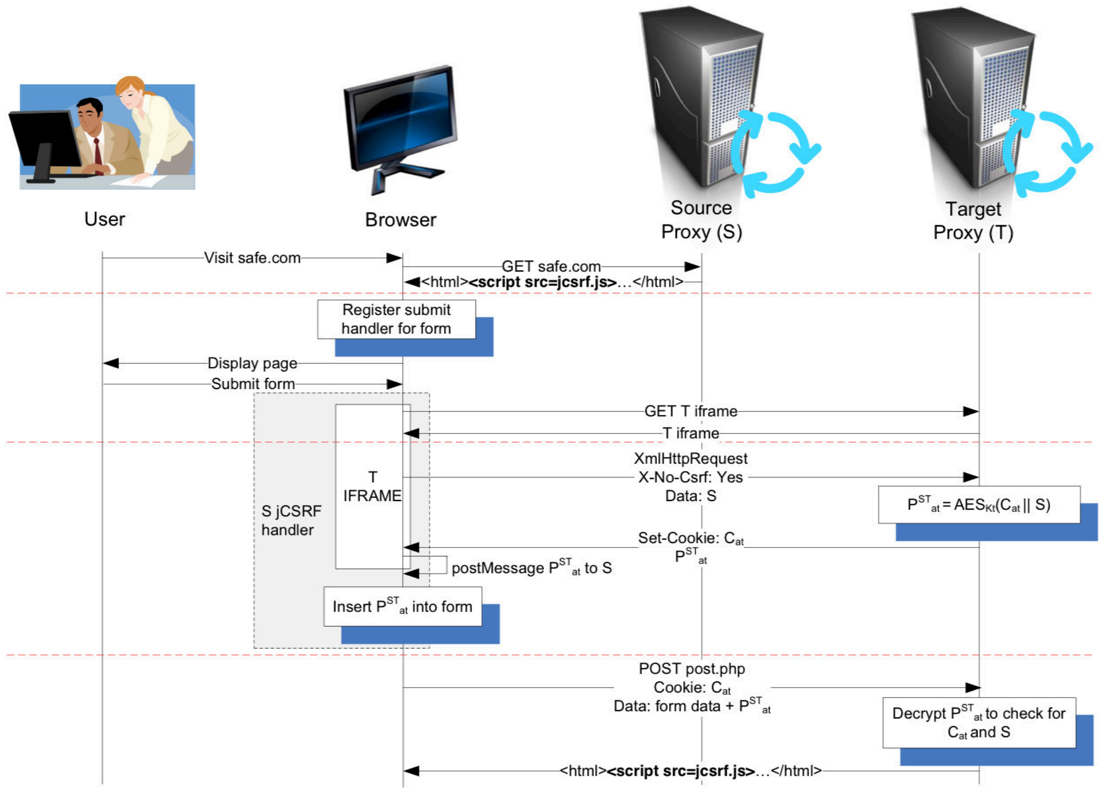

User
Browser
Source Proxy (S)
Target Proxy (T)
Visit safe.com
GET safe.com
<html><script src=jcsrf.js>...</html>
Register submit handler for form
Display page
Submit form
GET T iframe
T iframe
XmlHttpRequest X-No-Csrf: Yes Data: S
S jCSRF handler
T IFRAME
$P^{ST}_{at} = AES_{Kt}(C_{at} || S)$
Set-Cookie: C_{at} P^{ST}_{at}
postMessage P^{ST}_{at} to S
Insert P^{ST}_{at} into form
POST post.php Cookie: C_{at} Data: form data + P^{ST}_{at}
Decrypt P^{ST}_{at} to check for C_{at} and S
<html><script src=jcsrf.js>...</html>

Figure 3: Cross-Origin Protocol Workflow

generate a sequence of pseudorandom numbers $R_1, R_2, \ldots$. From these, nonces $N_1, N_2, \ldots$ are generated using secret-key encryption (specifically, the AES algorithm) as follows:

$$K_s = IR,\ N_i = AES_{K_s}(R_i)$$

Whenever jCSRF-proxy receives a request with a missing (or invalid) C_{at}, it sets C_{at} to N_i and increments i.

Note that this protocol design does not require N_i values to be stored persistently, since the validation check is stateless: jCSRF-proxy simply needs to compare C_{at} and P_{at} values in the submitted request. Hence, if jCSRF-proxy crashes, it can simply start all over, generating a new IR and so on. Similarly, K_s can be refreshed on a periodic basis by setting it to a new random value from `/dev/random`.

2.2.2 *Cross-Origin Protocol*

Figure 3 illustrates our protocol that enables pages from a (source) domain S to submit requests to a (target) domain T. Note that servers have been omitted to reduce the number of actors involved in the picture. Before describing the specifics of the protocol, note that the mechanism used in the same-origin case cannot be used for cross-origin requests: jCSRF-script runs on the source domain and therefore has no access to the target domain's cookies, which should contain the authentication token for requests to that domain. An obvious approach for overcoming this problem is to have the source domain communicate directly with the target domain to obtain its authentication token, but this is not easy either. In particular, a correct protocol must bind the subset of user's cookies containing security credentials (for domain T) to jCSRF's authentication token (also for domain T). Unfortunately, jCSRF-proxy, being application-independent, is unaware of which cookies contain user credentials, and hence cannot achieve such a binding on its own. We therefore develop a protocol that exploits browser-side functionality to avoid the need for a new protocol between S and T. In this protocol, javascript code executing on the user's browser communicates with T to obtain an authentication token and communicates it to jCSRF-script. This enables jCSRF-script to include the right value of P_{at} when it makes its cross-origin request to T.

Note that there may be many instances where the user loads a page from S containing a form for T, but never actually submits it. To avoid the overhead of additional communication with T in those instances, the steps for passing T's authentication token to jCSRF-script are performed only when the user submits a cross-domain form. In addition to reducing the overhead, this approach has privacy benefits since T does not get to know each time the user visits a page that allows submitting data to T.

The specifics of our cross-domain protocol are as follows. When a POST action is performed on a page from S, jCSRF-script checks if the target domain T is different from S and if T accepts authenticated requests. This information can either be supplied by the web administrator as a list of jCSRF-compatible origins or detected by attempting to load a special image `jCSRF-image.jpg` from T: the `error` and `load` events can be used to detect whether the resource was found. If the host does not support jCSRF, then jCSRF-script simply submits the post to T without any authentication tokens. Otherwise, jCSRF-script injects an iframe into the page for the URL `http://T/jCSRF-crossdomain.html?domain=S`. This page will contain javascript code that sets up the authentication token P^{ST}_{at} that a page from S can present to

T. The steps involved in this process are as follows:

- First, the script within the iframe makes an `XmlHttpRequest` to the domain T, providing S (given by the parameter `domain` in the above request) as an argument. `XmlHttpRequests` can only be issued to same-origin resources and, unlike ordinary requests, are allowed to include custom HTTP headers. Therefore, a request bearing the custom header `X-No-CSRF` proves to T that the request came from a page served to the user's browser by T.

- This `XmlHttpRequest` is served by jCSRF-proxy. If the user's jCSRF cookie (i.e., the cookie C_{at}) is not set, it is set by jCSRF-proxy using a nonce value N_i as described for the same origin case. In addition, jCSRF-proxy sends back the authentication token:

  $$P_{at}^{ST} = AES_{K_T}(C_{at}||S)$$

  Here, K_T is a (random-valued) secret key generated for T using the procedure described for the same-origin protocol.

- In the next step, P_{at}^{ST} needs to be passed on jCSRF-script so that it can complete the request to server T. This is accomplished using the `postMessage` API, which provides a secure mechanism for the framed script from domain T to communicate with a script from domain S. Note that a framing page from a malicious domain A cannot trick the frame from T into sending P_{at}^{ST}: `postMessage` can be instructed to deliver the message to a specific target origin which is chosen by T. Whenever T is instructed to send P_{at}^{ST}, this will be sent to S only, thus preventing A from reading the message.

  Some of the older browsers do not support the `postMessage` API. In that case, a technique called *location hash polling*[1] can be used in its place.

- Once the framing page has received the token, the jCSRF-script from S adds it to the form and submits the POST request to T.

- When jCSRF-proxy for domain T receives a POST request, it decrypts it using K_T, and checks if the cookie C_{at} included with the POST is a prefix of the decrypted data. If so, it checks if the domain S, which represents the remaining part of the decrypted data, is authorized to submit cross-domain POST requests. If so, the request is passed on the server. In all other cases, jCSRF-proxy treats the request as unauthorized, and strips all cookies before it is forwarded to T.

Correctness.

Correctness of the cross-domain protocol relies on the same three properties as the same origin protocol:

- scripts running on an attacker-controlled page visited by user's browser cannot obtain the authentication token for the protected domain.

- any token that may be obtained by the attacker, say, using his own browser, cannot be used to authenticate a request from user's browser to the protected domain

- the attacker should not be able to guess an authentication token that is valid for the protected domain

For the first property, note that because of the semantics of the `postMessage` API, an attacker-controlled page can either receive an authentication token that encodes its true domain A, or it may lie about its origin and not receive the token at all. In the latter case, the first property obviously holds. In the former case, although there is an authentication token, it contains the true origin of the attacker. On receiving this token, jCSRF-proxy will deny the request, as the attacker domain A is not authorized to make cross-origin posts.

The second property holds because the attacker is unable to set (or read) the value of user's cookie C_{at} for the protected domain T. Thus, even if he obtains an authentication token P by interacting with T using his own browser, he cannot use it with user's cookie C_{at} that will have a different value from the cookie value sent to the attacker by T. (Recall that T uses cryptographically random nonces to initialize C_{at}.)

The third property is ensured by the fact that the authentication token is randomly chosen from a reasonably large keyspace.

2.3 Design and Operation of jCSRF-script

As noted before, jCSRF-script needs to intercept all POST-requests and add the authentication token as an additional parameter to these requests. There are two ways in which browsers may issue POST requests:

- *Submission of HTML forms,* represented by `form` tags. Note that it is not necessary for the page to contain a `form` tag, because the form can be constructed dynamically using Javascript. Also, it is not necessary for the user to submit the form explicitly, because the form can be submitted automatically using Javascript.

- *`XmlHttpRequest` submissions.* Unlike form submissions, where the response cannot be accessed by the submitting script, the response to `XmlHttpRequest` can be read by the script making the request.

Compatibility requires handling both types of primitives. We now describe how jCSRF-script achieves this.

2.3.1 HTML Forms

Modern browsers allow Javascript code to register callbacks for specific events concerning the web page presented to the user. These functions are called *event handlers.* To ensure that every form is submitted to the web application with an authentication token, jCSRF-script registers a *submit* handler for each POST-based form. This handler then checks if the submission is to the same-origin or cross-origin. In the former case, jCSRF-script simply adds the authentication token as an additional parameter to the POST request. In the latter case, it uses the cross-domain protocol to first

[1]In location hash polling [10], a framing page sends its URL to a framed page as a parameter. The framed page can then append a token to the URL of the framed page using an anchor at the end of the URL. The basis for this technique is that this URL change does not cause the framing page to reload; instead the value appended to the URL is available for polling. If a malicious page from domain A lies about its URL (pretending to be a page from S), then the update will cause the outer page to reload from domain S, thus defeating the attempt by A to read data written by T.

obtain a token for the target domain, and then adds this token as an additional parameter to the POST request.

Note that the web application might define its own event handlers for the submission event, mostly to validate the form contents. If the web application handler ran after jCSRF's handler, it would have access to the authentication token. In some rare cases, the presence of the token might confuse the web application handler which only expects a predefined set of fields. Therefore, jCSRF-script detects if the web application defines its own handlers and wraps them with a function which removes the token before calling the web application handler, reinserting the token afterward. There are two types of event handlers: DOM0 handlers (registered with the HTML attribute `onSubmit` or by assigning a function to the JavaScript property `form.submit`) and DOM2 handlers (registered with the `addEventListener` function). The former type of handler is detected by periodically checking all forms for new, unwrapped submit handlers, which can be done through the previously mentioned `submit` property of form elements, since handlers are JavaScript functions and functions are first-class objects in JavaScript. The latter type requires overriding the `addEventListener` method to directly wrap new handlers during their registration, since there is no way to query the set of listeners registered for a specific (event, object) pair. Overriding `addEventListener` requires DOM prototypes[11] support, which is not available on old browsers (IE7 and older). On these browsers, only DOM0 events can be wrapped.

Note that failure to wrap an event handler does not necessarily imply a compatibility issue with jCSRF: most web applications define their own handlers to predicate on specific form fields, enforcing constraints such as "the field `age` must be a number". Only handlers that predicate on classes of fields might be incompatible with jCSRF on older browsers. For example, the constraint "all fields must be shorter than 10 characters" could create a problem for the token field if the handler is not wrapped.

2.3.2 *XmlHttpRequest*

For `XmlHttpRequests`, jCSRF-script modifies the `send` method of the class. For a browser supporting DOM prototypes [11], this can be done simply by substituting the `send` function, while on older browsers it is done by completely wrapping `XmlHttpRequest` functionality in a proxy object that hides the original class, and redirects all requests made by the web application to the proxy class.

Since these requests can only be issued to same-origin resources, a new protocol is not required for the client-side script to prove that it originated on the server. Instead, the script can prove that it is from the server by simply performing an operation that would not be permitted for a script from a different source. Our particular choice was to set a custom header, which is permitted only for same origin requests. Note that, while `XmlHttpRequest`s are only allowed to same-origin resources and are therefore safe, there would be no way for jCSRF-proxy to distinguish them from ordinary requests without this header.

3. EVALUATION

3.1 Compatibility

To verify that jCSRF is compatible with existing applications, we deployed popular open-source Web applications and accessed them through the proxy, checking for false positives by manually testing their core functionality. We tested jCSRF with the following applications: phpMyAdmin, SquirrelMail, punBB, Wordpress, Drupal, Mediawiki, and phpBB. As Figure 4 shows, these are complex web applications consisting of thousands of lines of code that would require substantial developer effort to audit and fix CSRF vulnerabilities. jCSRF was able to protect all applications without breaking their functionality in any way.

Note that these applications did not require cross-origin requests, and therefore the protocol could not be tested directly. However, we can argue that the cross-origin protocol is also compatible with existing web applications: the only two sources of incompatibilities in jCSRF occur when:

1. jCSRF-script fails to register its handler and a request is sent out unauthenticated.
2. jCSRF policies do not fully capture the interactions required among the web applications and cause policy violations.

For 1), we point out that the compatibility test from Figure 4 also proves that jCSRF-script correctly handles POST requests: with respect to compatibility, the actual logic implemented by the handler is largely irrelevant, as long as the handler executes correctly. Whether the code simply copies a value from the cookies or whether it runs an authentication protocol with the target server, this is transparent to the web application. For 2) we point out that the policy is only enforced for POST requests to other servers running jCSRF. Therefore, assuming that 1) does not happen and that the web developers has correctly configured the list of allowed origins, every same-origin request and every cross-origin request to a protected website will bear a valid authentication token and will thus be allowed.

The test were carried out using Google Chrome and Firefox. It is worth mentioning that jCSRF requires JavaScript enabled. If it is disabled, the requests are sent out unauthenticated, which results in false positives. For this reason, jCSRF does not work well with extensions such as NoScript [18] when these disable scripting functionalities on the protected website.

3.2 Protection

To test the protection offered by jCSRF, we selected 2 known CVE vulnerabilities and attempted to exploit them. The results are summarized in Figure 5.

First, we exploited the CVE-2009-4076 [4] vulnerability on the open source web mail application RoundCube [21]. Emails are sent using a POST request, but its origin is not authenticated. We built an attack page on an external website that fills out and submits an email message. jCSRF successfully blocked the attack, because the POST request was missing the authentication token. Second, we exploited the CVE-2009-4906 [5] vulnerability on the Acc PHP eMail web application. This vulnerability allows changing the admin password with a POST request from an external website. jCSRF was able to thwart this attack as well.

We limited our evaluation to two because the effectiveness of jCSRF does not need to be established purely through testing. Instead, we have provided systematic arguments as to why the design is secure against CSRF attacks. A secondary factor was that reproducing vulnerabilities is a very time-consuming task, and can be further complicated

Application	Version	LOC	Type	Compatible
phpMyAdmin	3.3.7	196K	MySQL Administration Tool	Yes
SquirrelMail	1.4.21	35K	WebMail	Yes
punBB	1.3	25K	Bulletin Board	Yes
WordPress	3.0.1	87K	Content-Management System	Yes
Drupal	6.18	20K	Content-Management System	Yes
MediaWiki	1.15.5	548K	Content-Management System	Yes
phpBB	3.0.7	150K	Bulletin Board	Yes

Figure 4: Compatibility Testing

Application	Version	LOC	Type	CVE	Stopped
RoundCube	0.2.2	54K	Webmail	CVE-2009-4076	Yes
Acc PHP eMail	1.1	3K	Mailing List Manager	CVE-2009-4906	Yes

Figure 5: Protection Evaluation

by difficulties in obtaining specific software versions that are vulnerable, and dependencies on particular configurations of applications, operating systems, etc.

Finally, two attacks are out of scope for a tool such as jCSRF, but should be mentioned for completeness: XSS attacks and same-domain CSRF attacks. XSS attacks can be used to break the assumption that same-origin scripts are under the control of the web developer, to issue token requests and leak results to the attacker, thus defeating the purpose of jCSRF. We point out that a successful XSS attack grants the attacker far more serious capabilities than the ability to craft requests on the victim's browser using his cookies. In fact, the attacker can simply steal the cookies directly and send authenticated requests as the victim from his own machine! To our knowledge, no other server-side CSRF defense can resist in case of an XSS attack. Same-origin CSRF attacks can be carried out by injecting a form in a server response and tricking the user into submitting it. jCSRF-script would add the correct authentication token, because it has no way to realize that the form present in the DOM tree was indeed supplied by the attacker [25].

3.3 Performance

In this section, we estimate the overhead imposed by jCSRF. A page embedding jCSRF-script issues three different type of requests to its target jCSRF-proxy:

1. GET requests. For these, the browser does not perform any special processing, and thus incurs no overhead. On the server-side, jCSRF-proxy only needs to generate a new token if the user does not have one already.

2. Same-Origin POST requests. Before the actual submission, jCSRF-script copies the authentication token C_{at} from the cookies to the form as P_{at}. Therefore, no overhead is introduced on the client, and the proxy only needs to check that $C_{at} = P_{at}$, which is an inexpensive operation.

3. Cross-Origin POST requests. The cross-origin protocol requires two additional GET requests for authentication: one to fetch the iframe from the target web application that requests the token and one for the actual `XmlHttp-Request` that fetches the token [2]. Therefore, this additional network delay dominates any other delay introduced by token generation and verification by the proxy. Although this overhead is non-negligible, we point out that cross-origin POST requests make up only a small fraction of HTTP requests [17], and therefore the delay due to these roundtrips is not likely to affect the overall user browsing experience.

[2]plus an additional request if jCSRF support has to be autodetected

We built a simple web application, deployed it locally and compared the response time of unprotected vs. protected same-origin and cross-origin POST requests. jCSRF protection incurred at most 2ms overhead. Clearly, the computational overhead is limited and is not perceived by the user.

4. RELATED WORK

4.1 Server-side Defenses

NoForge [16] was the first approach that used tokens to ascertain same-origin requests without requiring modifications to the application's source code. Implemented as a server-side proxy, NoForge parses HTML pages served by a web server, and adds a token to every URL referring to this server. It associates this token with the cookie representing the session id for the application. When a subsequent GET- or POST-request is received, it checks if this request contains the token corresponding to the session id. jCSRF is clearly influenced by NoForge, but makes several significant improvements over it:

- NoForge requires developer help to specify the name of the cookie containing the session id. Not only is this effort unnecessary in our approach, but it is also the case that our technique is compatible with alternative schemes for authentication, such as those that store authentication credentials in multiple cookies, or schemes that support persistent logins that, at different times, may be associated with different session ids.

  Moreover, NoForge needs to maintain server-side state in the form of valid (session id, token) pairs. In contrast, jCSRF does not maintain state, and is less prone to DoS attacks.

- jCSRF supports web sites where URLs are dynamically created by client-side execution of scripts.

- jCSRF supports cross-origin requests whereas NoForge can only protect same-origin requests. NoForge's approach does not easily extend to cross-origin case since it relies on a mapping between cookies and tokens on the server side. In the cross-origin case, the cookies that are visible to the origin and target domains are different, and so it is unclear how the states maintained on the two domains can be correlated.

An important difference between NoForge and jCSRF is that the former protects GET-requests as well. However, as discussed before, there are a number of difficulties in CSRF protection for GET-requests: inability to bookmark pages, need for developer effort to identify "landing pages" that do not need CSRF protection (which are not supported by NoForge), and so on. In the interest of providing a simple, fully automated solution, jCSRF protects only POST-requests.

Bayawak [14] can be thought of as extending NoForge to enforce a stronger policy: URLs in the web application are augmented with a special token not only to ensure that the request is same-origin, but also to constrain the order in which web pages can be visited. As such, it also protects against *workflow attacks* that aim to disrupt the session integrity by sending out-of-sync requests. This increased protection is obtained at the cost of additional programmer effort needed to specify permissible workflows. Bayawak does not address cross-origin requests or URLs that are dynamically created on the client-side.

X-PROTECT [25] is a server side defense that employs white-box analysis and source code transformation to overcome the shortcomings of other black-box approaches, namely their inability to protect against same-origin CSRF and their need to specify landing pages manually.

4.1.1 Developer Tools

Most web frameworks for rapid application development [8, 12, 20, 13, 24] include functionality to simplify CSRF protection, typically using a NoForge-like approach. For example, Django [8], a Python-based framework, provides CSRF protection for POST requests by requiring a specific template tag to be added to HTML forms, which is translated to a hidden form field containing a token that is also returned through cookies. To check whether the token in the cookies and the form match before executing the application logic, Django provides function wrappers to instrument *views* (python functions associated to URLs). CSRF-Magic [23] provides a similar capability for PHP applications. Web developers need to include an import statement in their PHP files to activate this protection. The purpose of the included file is to register output and input filters. The output filter executes before the HTML page is sent to the client, and adds a token to POST forms. The input filter checks for the presence of this token.

CSRFGuard [22] is similar to CSRFMagic, but is designed for Java EE applications. It examines incoming GET and POST requests for the presence of a valid token. CSRF-Guard introduced an option for client-side insertion of tokens using a script. Although this appears similar to our technique of injecting jCSRF-script, its operation is different. In particular, their client-side script adds the token to content available when the page fires the `load` event, and hence does not handle requests that may be dynamically constructed by various scripts associated with this page. Moreover, unlike NoForge, it allows web developers to configure a set of landing pages that do not require a valid token, thus mitigating the usability issues related to GET-request protection at the cost of additional developer effort.

CSRF tools for developers are an invaluable resource for rapid application development. Their benefit is that they provide a finer granularity of control for programmers, as compared to fully automated approaches such as jCSRF. The main drawback of developer tools is the need for programmer effort. Moreover, programmers may overlook to add checks in all places they are required, thus leaving vulnerabilities.

The basic idea of comparing a token and cookie value to verify same origin requests is similar between these approaches and jCSRF. However jCSRF goes beyond these tools by (a) providing support for cross-origin requests, and (b) supporting requests to URLs that are generated dynamically on the client-side.

4.2 Browser Defenses

Zeller and Felten [24] present a Firefox plugin which implements a simple policy to prevent POST-based CSRF: cross-site POST requests must be authorized by the user. The drawback of this simple approach is the fatigue stemming from repeated user prompts. NoScript [18] implements a more sophisticated policy that can avoid prompts. In particular, it restricts only those POST requests that go from an untrusted origin to a trusted origin. NoScript primarily targets sophisticated, security-conscious users who are willing to put in the effort needed to populate their list of trusted origins.

De Ryck et al [7] presents a CSRF protection plugin for Firefox, CsFire. It studies cross-domain interactions on the web, and uses the results to design a cross-domain policy that protects against CSRF attacks while optimizing compatibility with existing web applications. This policy relies on the concept of *relaxed SOP*, which allows communication between subdomains of the same registered domain (e.g. `mail.google.com` and `news.google.com`). Their policy also introduces the idea of *direct interaction*: since CsFire is a browser plugin, it has access to UI information such as whether a request was initiated by a user click. Cross-Origin GET requests initiated by user clicks are allowed, while cross-origin POST requests are not allowed in any case. Instead of blocking the request altogether, the plugins strips the cookies from the request, which are necessary to carry out a successful CSRF attack.

RequestRodeo [15] differs from the above techniques in that it is implemented outside of a browser as a client-side proxy. It relies on an approach similar to NoForge, but rather than blocking a request, it simply strips all cookies from such requests. Another difference is that it has no exceptions for landing pages. This can significantly affect usability.

A key advantage of browser-side defense is that it protects users even if web sites are not prompt in fixing their vulnerabilities. Moreover, they have accurate information about the origin of requests, whether they result from clicking on a bookmark, or a link on a web page trusted by a user. Their primary drawback is that the defense is applied to all web sites and pages, regardless of whether they have any significant security impact. Such indiscriminate application significantly increases the odds of false positives. Moreover, it is easier for a server-side solution residing on a target domain to determine whether it trusts the origin domain of a request. In contrast, browser-based defenses require the user to make this determination, and moreover, do it for all origins and target domains.

4.3 Hybrid Defenses

These defenses require both browser and server modifications. Referrer headers are the best known mechanism in this context. Using this HTTP header, a web browser can provide the crucial information that servers lack: namely, the origin domain of the current request. Given this information, a server can implement a simple CSRF protection mechanism that denies requests from domains that it does not trust. Unfortunately, due to privacy concerns, it is common to suppress referrer headers [1]. Barth et al [1] proposed a new header, called the *origin header*, to overcome these privacy concerns by suppressing some of the information provided by referrer headers, e.g., the query string. Currently, only Webkit-based browsers implement this header. Moreover, there may be lingering privacy concerns even with this origin header.

SOMA [19] is an alternate approach that aims to address a range of threats, including CSRF and XSS. With SOMA, a target domain is able to specify the set of allowable origin domains and vice-versa. Its implementation relies on a browser plug-in that retrieves these policies from the source and target domain and disallows any cross-origin requests that violate either policy.

Hybrid defenses often represent the best solutions that can be achieved by bringing together the information and mechanism that are available on browsers as well as webservers. Their key drawback is that both sides have to be modified simultaneously in order to achieve their benefits. For this reason, their adoption can take a long time. For instance, the origin header represents a relatively modest change, but even after about 3 years since that proposal was made, there is just a single major browser that supports it.

5. CONCLUSIONS

We introduced jCSRF, a tool to transparently protect web applications against CSRF attacks without requiring source code changes or configuration. Unlike similar solutions which modify all outgoing HTML responses to contain tokens, jCSRF uses JavaScript to augment requests dynamically. This allows jCSRF to also handle requests to resources that are not directly served by the protected web application, but rather generated dynamically on the browser. Moreover, jCSRF provides a protocol to authenticate cross-origin requests, which extends the applicability of the tool to complex, multi-domain deployments. Because it is implemented as a proxy, it should pose no compatibility problems with any web application, regardless of the language used or the web server it runs on. Our evaluation results show that jCSRF is a practical solution for automatically protecting web applications against CSRF attacks.

6. ACKNOWLEDGMENTS

We thank Siddhi Tadpatrikar for her work on implementing jCSRF-script for the same origin protocol.

7. REFERENCES

[1] A. Barth, C. Jackson, and J. C. Mitchell. Robust Defenses for Cross-Site Request Forgery. In *CCS*, 2008.

[2] CVE Editorial Board. CVE-2007-3574. `http://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2007-3574`, 2007.

[3] CVE Editorial Board. CVE-2009-2073. `http://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2009-2073`, 2009.

[4] CVE Editorial Board. CVE-2009-4076. `http://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2009-4076`, 2009.

[5] CVE Editorial Board. CVE-2009-4906. `http://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2009-4906`, 2009.

[6] CWE and SANS Institute. 2010 CWE/SANS Top 25 Most Dangerous Software Errors. `http://cwe.mitre.org/top25/`, March 2011.

[7] P. De Ryck, L. Desmet, T. Heyman, F. Piessens, and W. Joosen. CsFire: Transparent client-side mitigation of malicious cross-domain requests. In *ESSOS*, 2010.

[8] Django Software Foundation. Django. `http://www.djangoproject.com`, 2011.

[9] R. Fielding et al. Hypertext Transfer Protocol – HTTP/1.1. `http://www.ietf.org/rfc/rfc2616.txt`, 1999.

[10] J. Fraser. Backwards compatible window.postMessage(). `http://www.onlineaspect.com/2010/01/15/backwards-compatible-postmessage/`, 2010.

[11] F. Guisset. JavaScript-DOM Prototypes in Mozilla. `https://developer.mozilla.org/en/JavaScript-DOM_Prototypes_in_Mozilla`, 2002.

[12] D. H. Hansson. Ruby on Rails. `http://rubyonrails.org`, 2011.

[13] E. Inc. Code Igniter. `http://codeigniter.com/`, 2002.

[14] K. Jayaraman, G. Lewandowski, P. Talaga, and S. Chapin. Enforcing Request Integrity in Web Applications. *Data and Applications Security and Privacy*, 2010.

[15] M. Johns and J. Winter. RequestRodeo : Client Side Protection against Session Riding. In *OWASP Europe*, 2006.

[16] N. Jovanovic, E. Kirda, and C. Kruegel. Preventing cross site request forgery attacks. In *Securecomm*, 2007.

[17] W. Maes, T. Heyman, L. Desmet, and W. Joosen. Browser protection against cross-site request forgery. In *SecuCode*, 2009.

[18] G. Maone. NoScript. http://noscript.net/, 2011.

[19] T. Oda, G. Wurster, P. van Oorschot, and A. Somayaji. SOMA: Mutual approval for included content in web pages. In *CCS*, 2008.

[20] Pylons. Pylons Project. `http://pylonsproject.org/`, 2011.

[21] RoundCube.net. RoundCube - Free Webmail for the Masses. `http://roundcube.net/`, 2010.

[22] E. Sheridan. OWASP: CSRFGuard Project. `https://www.owasp.org/index.php/Category:OWASP_CSRFGuard_Project`, 2011.

[23] E. Z. Yang. CSRFMagic. `http://csrf.htmlpurifier.org/`, 2008.

[24] W. Zeller and E. Felten. Cross-site request forgeries: Exploitation and prevention, 2008.

[25] M. Zhou, P. Bisht, and V. Venkatakrishnan. Strengthening XSRF Defenses for Legacy Web Applications Using Whitebox Analysis and Transformation. In *ICISS*, 2011.

ASIDE: IDE Support for Web Application Security

Jing Xie, Bill Chu, Heather Richter Lipford, and John T. Melton
Department of Software and Information Systems
University of North Carolina at Charlotte
9201 University City Blvd
Charlotte, NC, 28223, USA
jxie2, billchu, heather.lipford@uncc.edu, jtmelton@gmail.com

ABSTRACT

Many of today's application security vulnerabilities are introduced by software developers writing insecure code. This may be due to either a lack of understanding of secure programming practices, and/or developers' lapses of attention on security. Much work on software security has focused on detecting software vulnerabilities through automated analysis techniques. While they are effective, we believe they are not sufficient. We propose to increase developer awareness and promote practice of secure programming by interactively reminding programmers of secure programming practices inside Integrated Development Environments (IDEs). We have implemented a proof-of-concept plugin for Eclipse and Java. Initial evaluation results show that this approach can detect and address common web application vulnerabilities and can serve as an effective aid for programmers. Our approach can also effectively complement existing software security best practices and significantly increase developer productivity.

Categories and Subject Descriptors

D.2 [**Software Engineering**]: [Software Security]; D.2.4 [**Software/Program Verification**]: [Correctness Proofs]; D.2.6 [**Programming Environments**]: [Integrated Environments]

General Terms

Security

Keywords

Secure programming, application security, interactive support, secure software development

1. INTRODUCTION

Software vulnerabilities originating from insecure code are one of the leading causes of security problems people face today [36]. Unfortunately, many developers have not been adequately trained in writing secure programs that are resistant from attacks violating program confidentiality, integrity, and availability. We refer to this concept of writing secure code as *secure programming* [4]. Programmer errors, including security ones, are unavoidable even for well-trained developers. One major cause of such errors is software developers' heavy cognitive load in dealing with a multitude of issues, such as functional requirements, runtime performance, deadlines, and security. Consider Donald Knuth's analysis of **867** software errors he made while writing TEX [19]. It is clear from the log that some of these errors could have made TEX vulnerable to security breaches. The following quote illustrates Knuth's experience of heavy cognitive burden as a major source of software errors:

> "Here I did not remember to do everything I had intended when I actually got around to writing a particular part of the code. It was a simple error of omission, rather than commission... This seems to be one of my favorite mistakes: I often forget the most obvious things" [19].

Current tool support for secure programming, both from tool vendors as well as within the research community, focuses on catching security errors after the program is written. Static and dynamic analyzers work in a similar way as early compilers: developers must first run the tool, obtain and analyze results, diagnose programs, and finally fix the code if necessary. Thus, these tools tend to be used to find vulnerabilities at the end of the development lifecycle. However, their popularity does not guarantee utilization; other business priorities may take precedence. Moreover, using such tools often requires some security expertise and can be costly. If programmers are removed from this analysis process, these tools will also not help prevent them from continuing to write insecure code.

We believe these vulnerability detection tools could be complemented by interactive support that reminds developers of good secure programming practices in situ, helping them to either close the secure programming knowledge gap or overcome attention/memory lapses. This approach can be justified based on cognitive theories of programmer errors [20, 21, 38]. Our hypothesis is that by providing effective reminder support in an IDE, one can effectively reduce common security vulnerabilities. Our approach is analogous to word processor spelling and grammar support. While people can run spelling and grammar checks after they have written a document, today's word processors also provide visual cues – colored lines drawn underneath potential errors – to help writers notice and fix problems while they are composing text. Similarly, our approach proposes that an IDE could interactively identify parts of the program where security considerations, such as input validation/encoding or Cross-site Request Forgery (CRSF) protection, are needed while programmers are writing code.

In this paper, we describe our interactive reminder approach through the **A**pplication **S**ecurity **IDE** (ASIDE), an Eclipse plugin for Java. ASIDE uses two key techniques to help programmers prevent errors: *interactive code refactoring* and *interactive code annotation.* While we have previously introduced this ***idea*** of an interactive approach to secure programming [37], in this paper, we present our prototype ***implementation***, along with a detailed ***evaluation*** of this plugin on two large-scale open source code bases, demonstrating its potential feasibility and effectiveness at identifying and preventing important security vulnerabilities. We also describe a user evaluation showing that users embrace such unobtrusive and helpful reminder support, and actively engage in the interaction with ASIDE. Thus, the major contribution of this paper is the multi-aspect evaluation of our proposed approach. Through our evaluation process, we also identify directions for future improvements to ASIDE and research.

2. ASIDE: APPLICATION SECURITY IN IDE

The ASIDE plugin is a prototype implementation of our interactive approach to reminding and helping programmers to perform good secure programming practices. Our approach was based on a number of design considerations. First, it is easiest and most cost effective for developers to write secure code and to document security implementation during program construction. This means creating a tool that integrates into the programmers' development environment is promising. As a starting point, we chose the popular Eclipse platform, and implemented ASIDE to work with Java. Our current implementation of ASIDE is primarily designed to assist Web-based applications and to mitigate commonly committed vulnerabilities due to improper input validation and/or filtering, broken access control, and Cross-site Request Forgery (CSRF). ASIDE, once activated, continuously monitors Eclipse workspace changes in order to respond to newly created projects, as well as modifications to existing projects. ASIDE performs static analysis incrementally on a small chunk of code under construction, thus ensuring prompt response to developers' immediate code editing. Such analysis is carried out "under the hood" without requiring developers to understand its security implications.

Our second consideration is the interface design principle that recognition is favored over recall [33]. Developers are provided with appropriate visual alerts on secure programming issues and offered assistance to practice secure programming. ASIDE provides these alerts as a colored icon on the left hand side margin of the code editing window, accompanied with highlighted text, see Figure 1. A developer can then either click on that icon or hover over the highlighted text to interact with ASIDE to address the issue. Our two key mechanisms for this interactive support are *interactive code refactoring* and *interactive code annotation*, which are discussed in detail in the following sections. There are no additional steps a developer must remember, and they can interact with ASIDE when they choose. Thus, the tool acts as a helpful assistant and does not hinder developers' creativity and productivity by dictating a rigid approach to secure programming.

Third, an in-situ reminder tool can be an effective training aid that either helps novices to learn secure programming practices or reinforces developers' secure programming training, making security a first class concern throughout the development process. This will help developers learn to reduce their programmer errors over time, reducing costly analysis and testing after implementation.

Finally, we want to support sharing secure programming knowledge and standards amongst development teams. In an industrial setting, ASIDE could be configured by an organization's software security group (SSG), which is responsible for ensuring software security as identified by best industry practice [11]. Thus, a SSG could use ASIDE to communicate and promote organizational and/or application-specific programming standards. In addition, the plugin can generate logs of how security considerations were addressed during construction, providing necessary information for more effective code review and auditing.

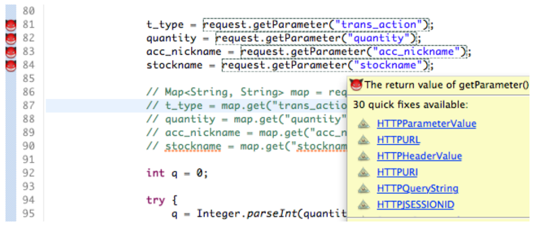
ASIDE first must detect potential vulnerabilities in program code, and alert programmers to those vulnerabilities. Thus, to be effective, ASIDE must exhibit a certain level of precision in identifying vulnerable code in order to provide proper assistance to address it. Based on our prototype implementation, a vulnerability can be easily eliminated by either of the two interactive techniques if the corresponding vulnerable code is correctly identified. Therefore, a key measure of the effectiveness of ASIDE is the measure of its ability to find vulnerable code. For each of the interactive techniques presented below, we present the details of how effective ASIDE is at detecting the corresponding vulnerabilities. The effectiveness of the interactive techniques to address those vulnerabilities heavily depends on how the programmer responds to the warning. This we evaluate based on a preliminary user study of programmer behaviors, presented in Section 7.

3. INTERACTIVE CODE REFACTORING

Interactive code refactoring [37] works in a manner similar to a word processor that corrects spelling and grammatical errors. In this case, ASIDE automatically inserts necessary code, with the help of the programmer. We believe code refactoring can be appropriately applied to input validation and/or filtering types of vulnerabilities. In this section, we discuss an example concerning input validation to illustrate the key concepts.

A developer is alerted by a marker and highlighted text in the edit window when input validation is needed. ASIDE has a rule-based specification language, which is also XML-based, to specify sources of untrusted inputs which we formally named as trust boundaries. Currently two types of rules are supported: Method (API) invocations, for example, method texttgetParameter(String parameter) in class `HttpServletRequest` introduces user inputs from clients into the system; and Parameter input, for instance, arguments of the Java program entrance method `main(String[] args)`.

With a mouse click, the developer has access to a list of possible validation options, such as a file path, URL, date, or safe text. Upon the selection of an option, appropriate input validation code will be inserted and the red marker will be dismissed. Figure 1 shows a screenshot of ASIDE facilitating a developer to select an appropriate input validation type for an identified untrusted input. The library of input validation options can be easily reconfigured by an individual developer or an organization.



```
        t_type = request.getParameter("trans_action");
        quantity = request.getParameter("quantity");
        acc_nickname = request.getParameter("acc_nickname");
        stockname = request.getParameter("stockname");

        // Map<String, String> map = req
        // t_type = map.get("trans_actio
        // quantity = map.get("quantity"
        // acc_nickname = map.get("acc_n
        // stockname = map.get("stocknam

        int q = 0;

        try {
            q = Integer.parseInt(quantit
```

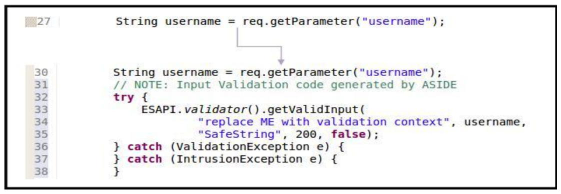
The return value of getParameter()

30 quick fixes available:

- HTTPParameterValue
- HTTPURL
- HTTPHeaderValue
- HTTPURI
- HTTPQueryString
- HTTPJSESSIONID

Figure 1: The user interactively chooses the type of input to be validated using a white-list approach.

Figure 2 illustrates how ASIDE refactors code to perform input validation using OWASP Enterprise Security API (ESAPI) Validator [28].



```
String username = req.getParameter("username");
```

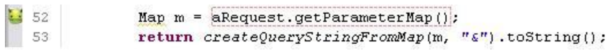
```
String username = req.getParameter("username");
// NOTE: Input Validation code generated by ASIDE
try {
    ESAPI.validator().getValidInput(
            "replace ME with validation context", username,
            "SafeString", 200, false);
} catch (ValidationException e) {
} catch (IntrusionException e) {
}
```

Figure 2: ASIDE validates an input using OWASP ESAPI validator API.

Previous works employing refactoring techniques for secure programming use program transformation rules, which operate on completed programs, and thus work best on legacy code. One recognized limitation of the program transformation approach is the lack of knowledge of specific business logic and context [12]. In contrast, our approach is designed to provide interactive support for secure programming and takes full advantage of developers' contextual knowledge of the application under development.

There are two possible strategies for when to perform input validation. One is to validate a variable containing untrusted input when it is used in a critical operation, such as a database update, insertion, or deletion. The other is to validate an untrusted input as soon as it is read into an application-declared variable. A major disadvantage of the first strategy is that it is not always possible to predict what operations are critical, and thus fails to validate input when the context of the application evolves. ASIDE promotes what we believe is the best practice for secure programming, validating untrusted inputs at the earliest possible time [8].

Another issue is that untrusted inputs could be of composite data type, such as a List, where input types may be different for each element of the list. In the current ASIDE implementation, the developer is warned of taint sources of a composite type with a visually softer yellow marker as shown in Figure 3. ASIDE uses data flow analysis to track an untrusted composite object. As soon as the developer retrieves an element that is of primitive data type (e.g.`java.lang.String`), ASIDE alerts the need to perform input validation and/or encoding in the same manner as described above. Given that the element retrieval from a composite data type is unbound, ASIDE leaves the marker shown in Figure 3 throughout the development to serve as a continual reminder.



```
        Map m = aRequest.getParameterMap();
        return createQueryStringFromMap(m, "&").toString();
```

Figure 3: Visually softer marker that marks a tainted input with composite data type: `Map`.

ASIDE supports two types of input validation rules: syntactic rules and semantic rules. A syntactic rule defines the syntax structure of an acceptable input and is often represented as a regular expression. Examples include valid names, addresses, URLs, filenames, etc. Semantic rules depend on the application context. For example, restricting the domain of URLs, files under certain directories, date range, or extentions for uploaded files. They can also be used to validate inputs of special data types, such as certain properties of a sparse matrix. While validation rules can be declaratively specified by a SSG, a developer has the option to address the identified input validation issue by writing custom routines. This is then documented by ASIDE for later security audit. A developer can also easily dismiss ASIDE warnings if they are determined to be irrelevant. In any case, once the alert has been addressed, the corresponding red marker and text highlights will disappear in order to reduce programmer distraction and annoyance.

Another benefit of ASIDE is that it can help enforce secure software development standards across the organization. For example, a company may deploy a validation and/or encoding library and clearly define a set of trust boundaries for the purpose of performing input validation and/or encoding. Once appropriatly configured with the defined trust boundaries and libraries, ASIDE can collect information as to where in the source code an untrusted input was brought into the system and what actions a developer took to address validation and/or encoding. ASIDE can also effectively supplement the security audit process by generating rules for traditional static analyzers. For example, once an untrusted input has been validated, customized Fortify rules can be generated to remove taints, thus avoiding unnecessary issues being generated during the auditing process. This can significantly reduce the time of a software security audit.

4. EVALUATION OF CODE REFACTORING

We now evaluate the effectiveness of ASIDE's vulnerable code detection for interactive code refactoring using an open source project of realistic scale. Thus, in this section, we focus on input validation and/or encoding vulnerabilities, as the ones currently supported by code refactoring. Our goals are to determine: (a) How effective is ASIDE at discovering exploitable software vulnerabilities and preventing them? and (b) What constitutes false positives for ASIDE? The significance of this evaluation is the use of real world cases that help us understand the effectiveness of ASIDE and provide guidance for further research.

4.1 Establishing a baseline using an open source project

We selected Apache Roller (a full-featured blog server) [30] release version 3.0.0 because it is one of the few mature Java EE based open source web applications. A Google search of "powered by Apache Roller" yielded over 1.8M entries including sites such as blogs.sun.com. One of the co-authors, who is an experienced member of a SSG at a large financial service organization, performed a software security audit using his company's practices to identify security vulnerabilities that are exploitable in Roller.

The audit process consisted of two parts: (1) automatic static analysis using Fortify SCA [34], and (2) manual examination of Fortify findings. Default Fortify rules were used, followed by manual auditing to eliminate Fortify findings that are not immediately exploitable. For each issue reported by Fortify, its source code was reviewed to:

- determine whether appropriate input validation/encoding has been performed;
- determine whether Fortify's environmental assumption is valid. For example, in the case of log forging, whether the logging mechanism has not been wrapped in any way that prevents log forging;
- determine whether Fortify's trust boundary assumption is valid. For instance, whether property files are considered to be trusted, and in this case, data from property files is untrustworthy;
- scrutinize input validation and encoding routines to make sure they are proper. For example, check if blacklist-based filtering is used. File, LDAP, DB, and Web all require different encoders or filters because different data schemes are used; and
- pay close attention to DOS related warnings (e.g. file handles and database connections) as resources may be released in a non-standard way. Often times, warning are generated even when resources are released in a `finally` block.

Roller 3.0.0 has over 65K lines of source code. Fortify reported **3,416** issues in **80** vulnerability categories, out of which, **1,655** issues were determined to be exploitable vulnerabilities. Table 1 summarizes the results of this audit process. Based on the evaluator's experience, Roller's security quality is at the average level of what he has evaluated. According to current work load estimate metrics of his enterprise, the analysis work reported here would amount to **2.5** person days.

Table 1: Results rendered by the industry security auditing process on Apache Roller version 3.0.0.

	Critical	High	Medium	Low
Fortify Issue Categories	8	18	2	52
Raw Issues	164	653	13	2,597
Exploitable Issues	37	397	0	1,221

ASIDE's code refactoring is primarily aimed at preventing vulnerabilities resulting from lack of input validation and/ or encoding. Out of the **1,655** Fortify issues that can be exploited in Roller, **922 (58%)** of them are caused by lack of input validation and/or encoding including most of the vulnerabilities from the critical bucket. The rest, mostly in the low security risk category, are related to failure to release resources (e.g. database connection) and other bad coding practices. Table 2 lists the details of the audit findings for the **922** issues of input validation and/or encoding we will compare to ASIDE.

It is common that multiple Fortify issues share the same root cause of an untrusted input, referred to as a *taint source*. A single taint source may reach multiple *taint sinks*, exploitable API calls, and thus generates several different vulnerabilities. For example, a user-entered value might lead to a Log Forging vulnerability if it is inserted into a log, and a SQL Injection if it is used in an operation that executes a dynamic SQL statement.

The **922** Fortify issues are caused by **143** unique taint sources including both primitive data types (e.g. `java.lang .String`) and composite data types (e.g. `java.util.Map`). Variables requiring output encoding are always the result of a taint source. Thus, we exclude them in our analysis to avoid duplications.

Table 2: Detail results from security auditing against Roller using Fortify SCA.

Severity	Category Name	
Critical	Cross-Site Scripting: Persistent	2
	Cross-Site Scripting: Reflected	2
	Path Manipulation	19
	SQL Injection	11
Medium	Cross-Site Scripting: Persistent	31
	Denial of Service	4
	Header Manipulation	52
	Log Forging	252
	Path Manipulation	6
Low	Cross-Site Scripting: Poor Validation	6
	Log Forging (debug)	531
	SQL Injection	3
	Trust Boundary Violation	3
Total		922

4.2 Vulnerability Coverage of ASIDE

We then imported Roller into an Eclipse platform that has ASIDE installed and configured. ASIDE identified **131** of the **143 (92%)** exploitable taint sources. The remaining **12** cases involve JSP files and Struts form beans. The current ASIDE implementation does not cover these cases, but they could be easily handled in future implementations.

41 of the **143** are taint sources of composite data returned from APIs such as `org.hibernate.Criteria.list()` and `javax.servlet.ServletRequest.getParameterMap()`. ASI -DE performs dataflow tracking within the method where untrusted input is read. When a primitive value (e.g. `java. lang.String`) is retrieved from the composite data structure instance, ASIDE will raise a regular warning and provide assistance to validate and/or encode that input, as described in Section 3.

While we successfully identified tainted inputs of composite data types in Roller, in many cases, developers did not use the elements in that data object within the immediate method. Since ASIDE only currently performs taint tracking within the immediate method declaration, future implementations of ASIDE will be expanded to support taint tracking for composite objects beyond the scope of the immediate method declaration, which would then alert the programmer to all these primitive data type uses.

Our analysis also raised the issue of delayed binding. An example of delayed binding is the access methods in a POJO (Plain Old Java Object). For example, `setBookTitle()` method of a Java class `Book.java` with a `bookTitle` attribute of `String` type. Binding of access methods to input streams can be delayed until after the program is completed. Thus, at the time of writing the program, there is no strong reason to believe the input is untrusted. After completion of the application, an integrator may bind an untrusted input stream directly to a POJO, making the application vulnerable.

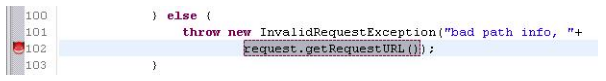
Delayed binding is a difficult problem for existing static analysis tools as well. If the binding specification (typically in `XML` format) is composed in the same IDE environment, which is usually the case for Java EE development, one could extend ASIDE to help developers discover input validation issues by resolving the binding specifications. Further research is needed on the best approach to address delayed bindings in ASIDE.

4.2.1 *False Positive for ASIDE*

As we just demonstrated, ASIDE had good coverage of the input validation/encoding issues in Roller. In this section, we discuss the additional warnings that ASIDE generated. In analyzing false positives, we only look at taint sources of primitive data types. Taint sources of composite types are accounted for when elements of the composite object are retrieved and treated as a taint source of a primitive data type.

ASIDE reported **118** taint sources of primitive data types that were not identified as exploitable Roller vulnerabilities by the Fortify software security audit. **94** of them are cases that are not exploitable at the moment. For example, a taint source does not reach any taint sink. Failure to validate/encode the untrusted input may not be exploitable in the context of the current application. However, often times, such untreated inputs will eventually be used, and thus cause exploitable security vulnerability as the software evolves. Therefore, we believe it is still a good secure programming practice to validate/encode all untrusted inputs, regardless of whether they will reach a taint sink or not.

Figure 4 shows another example from Roller, where a tainted request URL is directly passed into an `InvalidRequestException` constructor, and eventually inserted into the error log. Fortify default rules do not acknowledge this code to be vulnerable. However, if the logs are viewed in a web-based log viewer such as a web browser, which is common in some organizations, this would allow an attacker to launch a Cross-site Scripting attack on the system administrator reviewing the log.

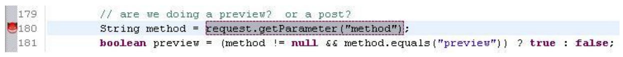


```
} else {
    throw new InvalidRequestException("bad path info, "+
                request.getRequestURL());
}
```

Figure 4: Untrusted input is logged through an Exception construction.

Thus, from a broad secure programming best practice perspective, we believe these **94** cases should be regarded as true positives, and ASIDE's warnings should still be followed. However, from a circumscribed perspective of a specific application, they may be regarded as false positives.

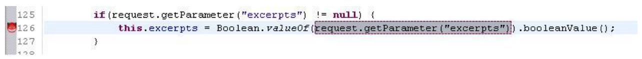
The remaining **24** reported taint sources we regard as false positive, where inputs are used in ways that do not lead to any recognized security vulnerabilities. These often involve inputs that are highly specific to the application context. For example, as illustrated in Figure 5, an input is tested to see if it equals to a constant value, determining the application flow.



```
// are we doing a preview?  or a post?
String method = request.getParameter("method");
boolean preview = (method != null && method.equals("preview")) ? true : false;
```

Figure 5: Untrusted input is used for logic test.

Another such case is shown in Figure 6, where the input is cast into a `Boolean` value with only two possible outcomes: true and false, which will not result in any harm to the intended application logic.



```
if(request.getParameter("excerpts") != null) {
    this.excerpts = Boolean.valueOf(request.getParameter("excerpts")).booleanValue();
}
```

Figure 6: Untrusted input is parsed into harmless `Boolean` value.

Because false positive rate often is positively correlated to accuracy, it is difficult to design a highly accurate tool with very low false positive rates. Both traditional analysis tools, such as Fortify SCA, and ASIDE will require manual inspection of warnings to eliminate false positives. However, our analysis of Roller suggests that for vulnerabilities due to improper input validation and/or encoding, ASIDE generates far fewer issues than Fortify, reducing the workload for both developers as well as software security auditors.

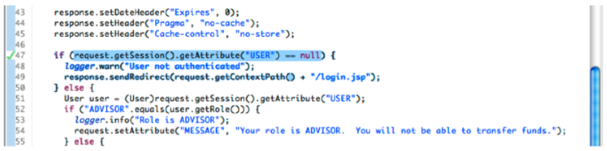
Additionally, we believe that it may take less effort to recognize and deal with ASIDE false positives compared to those generated by traditional static analysis. ASIDE's warnings are generated while the developer is actively engaged in the programming process, making it easier to examine and understand the context of the warning. Moreover, with a click of a button on ASIDE's resolution menu, the developer can dismiss a warning as false positive. In contrast, false positives generated by traditional analysis tools such as Fortify SCA are often dealt with by either software security auditors who typically do not have full application knowledge or by developers after the program was completed. In both cases, we believe it will take them longer to fully understand the impact of a particular warning generated by static analysis and to recognize it as a false positive. As excessive false positives could have a negative impact on the usability of any tool, further research is needed to understand how false positives in ASIDE impact developer behavior.

5. INTERACTIVE CODE ANNOTATION

Interactive code annotation [37] is a mechanism that helps developers to avoid more subtle security vulnerabilities where code refactoring is not feasible, such as broken access control and CSRF. Instead, programmers are asked to indicate where practices were performed, which is added as an annotation to the code. This serves as both a reminder to perform good practices, and enables further analysis and auditing. We first discuss interactive code annotation using an example of access control.

Consider an online banking application with three database tables: **user**, **account**, **account_user**, and **transaction**, where the tables **account** and **transaction** are specified as requiring authentication in such a way that the subject must be authenticated by the key of the **user** table, referred to as an access control table.

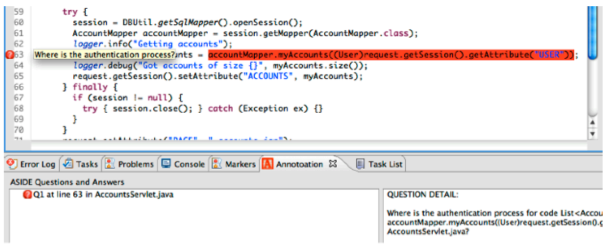
Figure 7(b) shows a highlighted line of code in the `doGet()` method of the accounts servlet, which contains a query to table **account**. ASIDE requests the developer to identify the authentication logic in the program, again by a red marker and highlighted text in the editing window. In this case, the developer highlights a test condition `reques.getsession().getAttribute(“USER”) == null` as illustrated in Figure 7(b), which ASIDE saves as an annotation to the query code. The annotations can be reviewed and modified in an additional view as shown in Figure 7(b) and Figure 7(c), on which different information corresponding to different actions the developer has taken is displayed. Thus, the annotation process is seamlessly integrated into the IDE without requiring the developer to learn any new annotation language syntax.



```
response.setDateHeader("Expires", 0);
response.setHeader("Pragma", "no-cache");
response.setHeader("Cache-control", "no-store");

if (request.getSession().getAttribute("USER") == null) {
  logger.warn("User not authenticated");
  response.sendRedirect(request.getContextPath() + "/login.jsp");
} else {
  User user = (User)request.getSession().getAttribute("USER");
  if ("ADVISOR".equals(user.getRole())) {
    logger.info("Role is ADVISOR");
    request.setAttribute("MESSAGE", "Your role is ADVISOR.  You will not be able to transfer funds.");
  } else {
```

(a) Developer identifies authentication logic (highlighted text) upon request from the ASIDE (see marker and highlighted text of Figure 7(b)) and annotates it.



```
try {
  session = DBUtil.getSqlMapper().openSession();
  AccountMapper accountMapper = session.getMapper(AccountMapper.class);
  logger.info("Getting accounts");
Where is the authentication process?nts = accountMapper.myAccounts((User)request.getSession().getAttribute("USER"));
  logger.debug("Got accounts of size {}", myAccounts.size());
  request.getSession().setAttribute("ACCOUNTS", myAccounts);
} finally {
  if (session != null) {
    try { session.close(); } catch (Exception ex) {}
  }
}
```

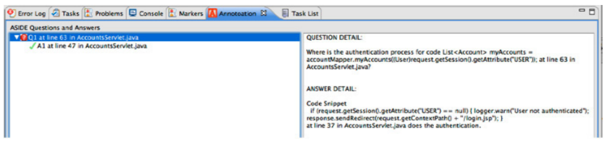
ASIDE Questions and Answers

Q1 at line 63 in AccountsServlet.java

QUESTION DETAIL:

Where is the authentication process for code List<Accou
accountMapper.myAccounts((User)request.getSession().g
AccountsServlet.java?

(b) ASIDE issues a question for proper access control check that grants/denies the access to the highlighted data access operation. The detail of such request is displayed on the view called *Annotation* below the code editing window.



ASIDE Questions and Answers

Q1 at line 63 in AccountsServlet.java
A1 at line 47 in AccountsServlet.java

QUESTION DETAIL:

Where is the authentication process for code List<Account> myAccounts = accountMapper.myAccounts((User)request.getSession().getAttribute("USER")); at line 63 in AccountsServlet.java?

ANSWER DETAIL:

Code Snippet
```
 if (request.getSession().getAttribute("USER") == null) { logger.warn("User not authenticated");
response.sendRedirect(request.getContextPath() + "/login.jsp"); }
```
at line 37 in AccountsServlet.java does the authentication.

(c) The *Annotation view* adds the annotated information as a response to the developer’s annotating of the access control logic in above Figure 7(a).

Figure 7: An example showing how ASIDE interactive code annotation works.

Several benefits can be derived from this annotation mechanism. First, the developer is reminded of the need to perform authentication, and the annotation process may help the developer to verify that authentication logic has been included. The developer has an opportunity to add intended access control logic should that be missing. Second, the logged annotations provide valuable information for code review. Third, heuristics-based static analysis can be performed to provide more in-depth analysis of the authentication logic.

We are still in the process of implementation interactive code annotation in ASIDE. Thus, the screenshots in this section are of our design, not a working prototype. We first discuss the issues in implementing code annotation, before moving on to our analysis of its potential effectiveness.

Our implementation of code annotation will take a knowledge-based approach relying on the specific structure of the target technology, initially Java servlet-based applications. We are first focusing on requesting programmers’ annotations on access control logic The key implementation issues are (a) how to identify the application context to prompt the developer for annotation and (b) what are the forms of acceptable annotations.

If we know the names of database tables whose access requires authentication, we can easily identify program statements that access these tables. Thus, we will provide a method for designers or a SSG to specify such tables as part of configuring ASIDE. However, such database access statements may not convey important application context, as they may be part of a utility library used by different web requests. For example, the **account** table may be accessed by the same access routine in multiple web requests (e.g. “customer account balance”, and “electronic fund transfer”). Also, different web requests may require different access control logic (e.g. two-factor authentication for fund transfer). Therefore, we need to identify locations where application logic takes place by invoking database access operations.

For Java servlet-based web applications, access control annotations can be requested in the context of a web request: as in “where is the authentication logic for accessing the account table in the customer account balance request?”, and “where is the authorization logic for accessing the account table in the electronic fund transfer request?” In this example each web request is implemented as a servlet. Therefore, we will start by tracking `doGet()` and `doPost()` methods within each servlet class (and this can be easily extended to include JSP pages)[1] . They are referred to hereafter as *entry methods*. Through static analysis techniques, we will detect cases where an entry method leads to a statement accessing a database table which requires access control. The developer will then be requested to provide annotation of authentication in the context of the entry method, as illustrated in Figure 7(b).

Our annotation design satisfies the following requirements: (a) it consists of a set of logical tests (e.g. conditional tests in `if` statement, cases in `switch` statements), as shown in Figure 7(a), and (b) each test must be on at least one execution path, as determined by the control flow, from the start of the entry method to the identified table access statement.

Annotations may enable more in-depth static analysis. We are specifically looking into one type of execution analysis. For example, a broken access control may be detected if there is an execution path in the entry method leading to the database access without any identified access control checks along the path. We believe such an analysis can also be used to help prevent CSRF vulnerabilities. Of course, the accuracy of this analysis depends on the accuracy of the annotation. We now move to our evaluation of code annotation against real world cases.

[1]Most popular Java frameworks also have structured entry methods, such as controller in Spring MVC

6. EVALUATE INTERACTIVE CODE ANNOTATION

We have conceptually tested this approach on real world open source projects. The security audit did not identify any broken access control or CSRF issues in Roller. Thus, we turned to bug tracking records to uncover previously discovered issues. Since Apache Roller only has a small number of fully documented security patches, we also included security patch information from Moodle [25], a PHP-based open source project for course management. A Google search of "powered by Moodle" yielded over **4.3M** sites including many large universities. A total of **20** fully documented security patches were found for the two projects. Four of them are due to improper input validation and/or encoding and can be addressed by ASIDE's code refactoring support. Out of the remaining **16** vulnerabilities, **3** (1 broken access control and 2 CSRF) can be addressed by code annotation and the path analysis heuristics outlined above.

The broken access control issue is from Roller [31]. The authenticator, as illustrated in Figure 8, gets a web request from the client side and checks to see whether the headers of the request are valid. If they are valid, it extracts the credentials and verifies the validity of them. If the credentials are valid, the program goes on to access protected data in the database. If the credentials are not valid, an exception will be thrown, thus preventing unauthenticated access. However, there is another path from the web entry point to the data access point when the headers do not conform to the expected format, as shown in the control flow diagram in Figure 8.

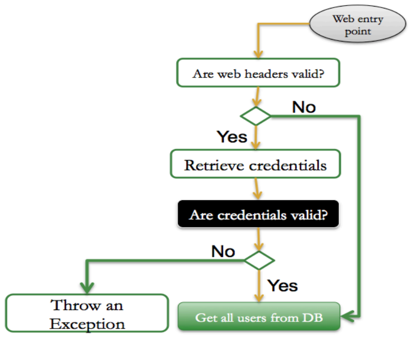
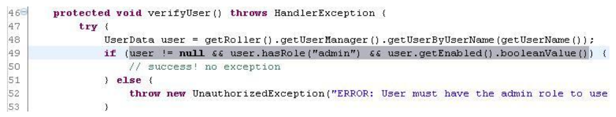


Figure 8: Control flow diagram of how an authentication request is processed.

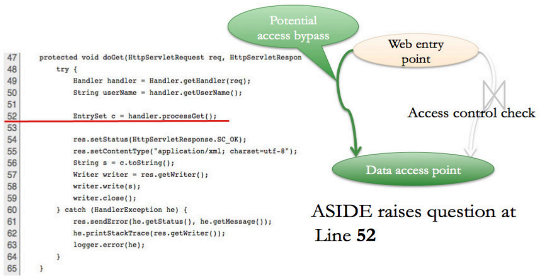
Applying ASIDE's code annotation approach, when the application code accesses the protected database resource to get all users' information, ASIDE would prompt a request for proper access control logic on the path from the web request to the data access method call. Considering that the question should be raised on a transaction level, line 52 in the Servlet processing the request would be highlighted, as shown in Figure 10. In this case, the developer could easily identify the access logic as the logic tests which lie in the method invocation `verifyUser(username, password)` in `BasicAuthenticator.java`, highlighted in Figure 9. In this case, there are three tests to be annotated.

Based on the developer's annotation, ASIDE would be able to construct a control flow graph, as illustrated in Figure 10, that has one path from a web entry point to a data access point with an annotated access control check on it, while another path from the same entry point to the same data access point has no access control check on it. Therefore, ASIDE would be able to provide a warning to the developer of a potential broken access control vulnerability.



```
protected void verifyUser() throws HandlerException {
    try {
        UserData user = getRoller().getUserManager().getUserByUserName(getUserName());
        if (user != null && user.hasRole("admin") && user.getEnabled().booleanValue()) {
            // success! no exception
        } else {
            throw new UnauthorizedException("ERROR: User must have the admin role to use
        }
```

Figure 9: Annotate access control logics.



```
47  protected void doGet(HttpServletRequest req, HttpServletRespon
48      try {
49          Handler handler = Handler.getHandler(req);
50          String userName = handler.getUserName();
51
52          EntrySet c = handler.processGet();
53
54          res.setStatus(HttpServletResponse.SC_OK);
55          res.setContentType("application/xml; charset=utf-8");
56          String s = c.toString();
57          Writer writer = res.getWriter();
58          writer.write(s);
59          writer.close();
60      } catch (HandlerException he) {
61          res.sendError(he.getStatus(), he.getMessage());
62          he.printStackTrace(res.getWriter());
63          logger.error(he);
64      }
65  }
```

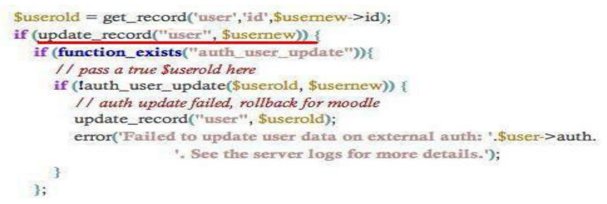
Figure 10: Java Servlet code for processing authentication request (left) and Access control check graph that involves processing the request (right).

Two CSRF vulnerabilities were recorded in Moodle's bug tracking system. An effective way to prevent CSRF is to assign a random token to each instance of a web page that changes the state of the application. The token will be verified by the server before the application state is changed. The Moodle project is clearly well aware of the CRSF problem and implemented this strategy as a set of standard utility functions, simplifying developers' tasks. However, developers still missed using this CSRF protection, introducing serious vulnerabilities into the software.

Through code annotation, ASIDE can be designed to remind developer of places where CSRF protection is needed, such as web transactions that change application states. Whenever a form submission/web request contains an operation to update (add, delete, modify) database entries, the form submission needs to be checked for CSRF. We describe one of the CSRF cases in Moodle, MSA-08-0013 [26], in detail; the other example is similar. This CSRF vulnerability is based on editing a user's profile page, *edit.php*. Since ASIDE is currently being designed for the Java EE environment, we recast this example in equivalent Java terms.

The edit function would have a web entry point such as in a Servlet. Function `update_record()` is called, as highlighted in Figure 11, to update selected database tables through database operations. ASIDE would prompt the developer to annotate a logic test that implements CSRF protection. The request for annotation would be made at the line where `update_record()` is called. In this case, CSRF protection was omitted, so the programmer would be reminded to add such protection. Once the CSRF protection is added with appropriate annotation, ASIDE will apply the path analysis heuristics to further check for possible logic errors that may bypass the CSRF protection.

We have not yet conducted an analysis on false positives



```
$userold = get_record('user','id',$usernew->id);
if (update_record("user", $usernew)) {
    if (function_exists("auth_user_update")){
        // pass a true $userold here
        if (!auth_user_update($userold, $usernew)) {
            // auth update failed, rollback for moodle
            update_record("user", $userold);
            error('Failed to update user data on external auth: '.$user->auth.
                        '. See the server logs for more details.');
        }
    };
```

Figure 11: A snippet of source code of the web transaction that changes user profile.

for code annotation since we have not completed a prototype of ASIDE's code annotation functions, however, we would like to point out a case where false positives may appear. It is not uncommon for a web application to access different tables of a database to respond to one single request. Without context, ASIDE would raise the same questions for each and every function call that changes a table, and thus may produce false positives. Further research is needed to make ASIDE more contextual and intelligent about asking questions in these cases.

7. DEVELOPER BEHAVIOR STUDY

The previous evaluations focused on the ability of ASIDE to detect or fix vulnerable code. However, ASIDE must be designed in a way that fits naturally into a developer's work environment in order to be successful. To gain an understanding of programmers' reactions towards real-time secure coding support, we conducted a pilot user study involving **9** graduate students from our college using our current implementation of ASIDE in a controlled experimental setting. All **9** participants were recruited from a Java based web application development course in the Spring 2011 semester. As part of the course, students were briefly introduced to basic secure web programming techniques such as input validation and encoding. However, project grades were assigned only based on functional requirements, not on secure coding practices.

Part of the students' course work was to build an online stock trading system incrementally over 4 projects throughout the semester. Our study focused on the last increment of this project where students were asked to implement functionality including add a banking account, display stock details, make a buy/sell transaction, and display transaction history. Students added these functions on top of their existing work artifacts, which included static web pages, login, logout, and register functionalities.

We asked students to come to our lab and work on their assignment for 3 hours, using Eclipse with ASIDE. ASIDE was implemented as described in Section 3, generating input validation and encoding warnings. Participants were given a brief tutorial of ASIDE, then told to work on whatever aspects of their code they wanted, using ASIDE as they wished. We recorded all their interactions with ASIDE through screen recording software, and logged all interactions with the ASIDE interface. Immediately after the 3-hour study, participants were interviewed about their experiences of using ASIDE. Participation was voluntary and not related to class grades; students were compensated with a $30 gift card for his/her time.

Our goals with this study were to evaluate the usability of ASIDE and determine: (a) do users pay attention to warnings? (b) do they use code refactoring functions, and (c) do warnings and code generation help developers produce more secure code?

Figure 12 depicts the results of all 9 participants. Over all 9 participants, **101** distinctive ASIDE warnings were generated, resulting in **11.2** warnings per participant on average. **83** warnings were clicked by participants, **9.2** for each participant on average.

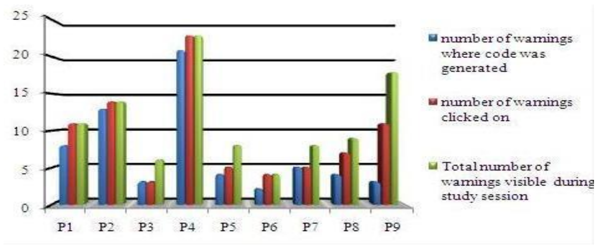


Figure 12: Metrics and results from 9 participants.

Out of the **83** warnings clicked, **63** were addressed (7 per participant on average) by clicking on one of the validation rules provided by ASIDE, leading to validation/encoding code being generated. The remaining warnings were dismissed by participants. All participants used ASIDE to generate validation routines in their development. None of them wrote any customized validation or encoding routines. Thus, ASIDE's code refactoring was effective in helping students write more secure code, even though they were not required to do so.

Multiple factors explain why certain warnings were not clicked or acted upon. Some of the warnings were generated when the participant wrote debugging code, which was soon deleted. Perhaps students ignored security warnings on code that they knew was transient. Other cases have to do with a bug in the version of ASIDE used, which falsely warned participants of the need for output encoding. We noticed that participants learned this warning was a false positive after one or more encounters, and then ignored those warnings thereafter. However, at least in this study, the presence of a false positive did not seem to cause the participants to not pay attention to other ASIDE warnings. Thus, in cases where participants encountered false positives, they were able to recognize them quickly and use the options provided by ASIDE to dismiss the false positives.

We transcribed all interviews into text via Inqscribe [15], and coded the transcripts using Atlas.ti [2] to identify general themes and interesting cases. All but one of the participants indicated that if they were not given ASIDE, they would not have been aware that some of the inputs needed to be validated before being used. Furthermore, they unanimously agreed that they would not write their own validation routines should ASIDE not provide them in the context of their assignment. As one participant stated:

> [P1] That (The warning design of ASIDE) was good because hadn't it prompted me, I wouldn't have to realize I have to inspect those input values.

No one expressed that the ASIDE warnings were annoying, and all seemed to have faith that using ASIDE would make their code more secure.

[P4] No (it does not bother me). It gives me warnings so that I can write secure code.

So overall, 8 out of 9 participants expressed a positive impression of ASIDE, with one being neutral. This study demonstrated that ASIDE was usable by our participants. They were able to utilize ASIDE quickly, in the context of their own development, potentially improving the security of their code. We plan to expand upon this study to examine the use of ASIDE by professional developers to more deeply investigate the impact on programmer behavior in a variety of contexts.

8. RELATED WORK

Research into tool support for software security focuses heavily on machine-related issues, such as technique advancements for vulnerability detection effectiveness, accuracy, and vulnerability coverage, with very little concern with human factors issues. The two prominent techniques are static and dynamic program analyses. Static analysis typically is based on taint tracking [27, 17, 21, 7] and dynamic analyses are often based on model checking [14, 9, 22] and symbolic execution [6, 10, 32]. As both approaches have their advantages and disadvantages, a variety of work has explored the combination of these two techniques in an attempt to achieve better performance [3, 23, 13] .

Software developer education and training has also been directed at achieving better software security. Most efforts have been spent on developing educational material and guidelines for the best secure programming practices [5, 16, 1, 18, 35]. However, the mere existence of such abundant information does not guarantee its use by programmers [38]. Our work takes human factors into consideration and fills this missing gap, complementing program analysis tools to help developers write more secure code.

Our work is in part motivated by studies on human errors [29], and programmer errors in particular [20], which demonstrate that programmer errors contribute a great deal to software flaws. Many such errors are caused by three types of cognitive breakdowns: *skill-based breakdowns* where programmers fail to perform routine actions at critical times; *rule-based breakdowns* where programmers fail to do an action in a new context; and *knowledge-based breakdowns* where a programmer's knowledge is insufficient. We believe that ASIDE's contextualized reminders can help address many of these issues by filling in knowledge gaps and reminding programmers of secure programming issues within their current context.

Prior work on code annotation (e.g. [24]) used textual extensions of programming languages, such as `C` and `Java`, which demands developers to learn and use yet another type of language. In contrast, our approach leverages GUI support to make the process more intuitive, direct and easy to use. Furthermore, soliciting security information from the developer through annotation could open up new ways to detect software vulnerabilities.

9. CONCLUSIONS

The main contribution of our work is to augment IDE tools to intelligently recognize software security issues and remind and assist developers with possible mitigation actions. We evaluated this approach against mature open source projects. Our results suggest that this approach is effective in detecting and helping prevent common types of web application vulnerabilities, such as lack of proper input validation and/or encoding, CSRF, and broken access control.

No single tool is capable of detecting all software vulnerabilities, and ASIDE is no exception. However, we believe that ASIDE can be most effective in supplementing static analysis tools and increasing the productivity of current best software security practices. For input validation and encoding issues, which often constitute the largest percentage of issues in a typical web application, ASIDE can significantly reduce the number of issues generated by static analysis by helping programmers to prevent them in the first place. Thus, ASIDE can help improve software security and reduce the number of security fixes needed after static analysis, as well as saving time for the software security audit by **50%**, based on our Roller case study.

Our user study results suggest that ASIDE is effective in helping novice developers/students to write more secure code when using code refactoring. They appear to pay attention to ASIDE warnings and follow ASIDE advice to perform input validation and/or encoding. Further studies, including studies involving experienced developers, are needed to show whether ASIDE can improve developers' understanding and practice of secure programming in more contexts. Preliminary evidence appears to suggest that users can quickly recognize false positives in ASIDE and take easy action to dismiss them. More research on false positives and how they impact on developers using ASIDE is needed.

Finally, our approach also provides a tool platform to provide additional support for the secure software development lifecycle in the areas of enforcing standards, information collection for secure coding metrics, and capturing developer rationale for code review or in depth program analysis. In an enterprise environment, ASIDE can serve as a medium that communicates a Software Security Group's secure coding knowledge to developers. As we continue the development of ASIDE, we plan to further investigate the use of these features and how they can contribute to secure applications.

ACKNOWLEDGMENTS This work is supported in part by a grant from the National Science Foundation 0830624, and a research/educational license from HP Fortify Inc. We would like to thank Will Stranathan for his contribution in the formation of this idea and critiques and suggestions on the intial prototyping of ASIDE. We also want to thank all our participants for their time.

10. REFERENCES

[1] S. Ardi, D. Byers, P. H. Meland, I. A. Tondel, and N. Shahmehri. How can the developer benefit from security modeling? In *The Second International Conference on Availability, Reliability and Security, 2007*, pages 1017 –1025, april 2007.

[2] Atlas.ti. Atlas.ti, 2011. `www.atlasti.com`.

[3] D. Balzarotti, M. Cova, V. Felmetsger, N. Jovanovic, E. Kirda, C. Kruegel, and G. Vigna. Saner: Composing static and dynamic analysis to validate sanitization in web applications. In *Proceedings of the 2008 IEEE Symposium on Security and Privacy*, pages 387–401. IEEE Computer Society, 2008.

[4] M. Bishop and B. J. Orvis. A clinic to teach good programming practices. In *Proceedings from the Tenth*

Colloquium on Information Systems Security Education, pages 168–174, June 2006.

[5] CERT. CERT Secure Coding, 2011. `www.cert.org/secure-coding`.

[6] A. Chaudhuri and J. S. Foster. Symbolic security analysis of ruby-on-rails web applications. In *Proceedings of the 17th ACM conference on Computer and communications security*, CCS '10, pages 585–594. ACM, 2010.

[7] B. Chess and G. McGraw. Static analysis for security. *IEEE Security and Privacy*, 2:76–79, November 2004.

[8] B. Chess and J. West. *Secure programming with static analysis*. Addison-Wesley Professional, first edition, 2007.

[9] V. Felmetsger, L. Cavedon, C. Kruegel, and G. Vigna. Toward automated detection of logic vulnerabilities in web applications. In *Proceedings of the 19th USENIX conference on Security*, USENIX Security'10, pages 10–10. USENIX Association, 2010.

[10] X. Fu and K. Qian. Safeli: Sql injection scanner using symbolic execution. In *Proceedings of the 2008 workshop on Testing, analysis, and verification of web services and applications*, TAV-WEB '08, pages 34–39. ACM, 2008.

[11] B. C. G. McGraw and S. Migues. Building security in maturity model, 2011. `www.bsimm2.com`.

[12] M. Hafiz, P. Adamczyk, and R. Johnson. Systematically eradicating data injection attacks using security-oriented program transformations. In *Proceedings of the 1st International Symposium on Engineering Secure Software and Systems*, ESSoS '09, pages 75–90. Springer-Verlag, 2009.

[13] Y.-W. Huang, F. Yu, C. Hang, C.-H. Tsai, D.-T. Lee, and S.-Y. Kuo. Securing web application code by static analysis and runtime protection. In *Proceedings of the 13th international conference on World Wide Web*, WWW '04, pages 40–52. ACM, 2004.

[14] Y.-W. Huang, F. Yu, C. Hang, C.-H. Tsai, D. T. Lee, and S.-Y. Kuo. Verifying web applications using bounded model checking. In *Proceedings of the 2004 International Conference on Dependable Systems and Networks*, pages 199–, Washington, DC, USA, 2004. IEEE Computer Society.

[15] Inqscribe. Inqscribe, 2011. `www.inqscribe.com`.

[16] S. Institute. SANS Institute, 2011. `www.sans.org`.

[17] N. Jovanovic, C. Kruegel, and E. Kirda. Pixy: a static analysis tool for detecting web application vulnerabilities. In *Security and Privacy, 2006 IEEE Symposium on*, pages 6 pp. –263, may 2006.

[18] K. Karppinen, L. Yonkwa, and M. Lindvall. Why developers insert security vulnerabilities into their code. In *Proceedings of the 2009 Second International Conferences on Advances in Computer-Human Interactions*, ACHI '09, pages 289–294. IEEE Computer Society, 2009.

[19] D. E. Knuth. The errors of tex. *Softw. Pract. Exper.*, 19:607–685, July 1989.

[20] A. J. Ko and B. A. Myers. A framework and methodology for studying the causes of software errors in programming systems. *J. Vis. Lang. Comput.*, 16:41–84, February 2005.

[21] V. B. Livshits and M. S. Lam. Finding security errors in Java programs with static analysis. In *Proceedings of the 14th Usenix Security Symposium*, pages 271–286, August 2005.

[22] M. Martin and M. S. Lam. Automatic generation of xss and sql injection attacks with goal-directed model checking. In *Proceedings of the 17th conference on Security symposium*, pages 31–43. USENIX Association, 2008.

[23] M. Martin, B. Livshits, and M. S. Lam. Finding application errors and security flaws using PQL: a program query language. In *OOPSLA '05: Proceedings of the 20th annual ACM SIGPLAN conference on Object oriented programming systems languages and applications*, pages 365–383, 2005.

[24] Microsoft. Microsoft SAL Annotations, 2011. `http://msdn.microsoft.com/en-us/library/ms235402.aspx`.

[25] Moodle. Moodle, 2011. `http://moodle.org`.

[26] Moodle. MSA-08-0013, 2011. `http://moodle.org/mod/forum/discuss.php?d=101405`.

[27] G. Naumovich and P. Centonze. Static analysis of role-based access control in j2ee applications. *SIGSOFT Softw. Eng. Notes*, 29:1–10, September 2004.

[28] OWASP. ESAPI Validator API, 2011. `http://owasp-esapi-java.googlecode.com/svn/trunk\_doc/latest/org/owasp/esapi/Validator.html`.

[29] J. Reason. *Human Error*. Cambridge University Press, Cambridge, UK, 1990.

[30] A. Roller. Apache Roller, 2011. `http://roller.apache.org`.

[31] A. Roller. ROL-1766, 2011. `https://issues.apache.org/jira/browse/ROL-1766`.

[32] P. Saxena, D. Akhawe, S. Hanna, F. Mao, S. McCamant, and D. Song. A symbolic execution framework for javascript. Technical Report UCB/EECS-2010-26, EECS Department, University of California, Berkeley, Mar 2010.

[33] H. Sharp, Y. Rogers, and J. Preece. *Interaction Design: Beyond Human-Computer Interaction*. Wiley, 2 edition, 2007.

[34] F. Software. Fortify SCA, 2011. `https://www.fortify.com/products/fortify360/source-code-analyzer.html`.

[35] B. Taylor and S. Azadegan. Moving beyond security tracks: integrating security in cs0 and cs1. In *Proceedings of the 39th SIGCSE technical symposium on Computer science education*, SIGCSE '08, pages 320–324. ACM, 2008.

[36] VERACODE. State of Software Security Report Volume 1, 2, and 3, 2011. `http://www.veracode.com/reports/index.html`.

[37] J. Xie, B. Chu, and H. R. Lipford. Idea: interactive support for secure software development. In *Proceedings of the Third international conference on Engineering secure software and systems*, ESSoS'11, pages 248–255. Springer-Verlag, 2011.

[38] J. Xie, H. R. Lipford, and B. Chu. Why do programmers make security errors? In *Proceedings of 2011 IEEE Symposium on Visual Languages and Human Centric Computing*, pages 161–164, 2011.

Tracking Payment Card Data Flow Using Virtual Machine State Introspection

Jennia Hizver
Department of Computer Science
Stony Brook University
jhizver@cs.stonybrook.edu

Tzi-cker Chiueh
Department of Computer Science
Stony Brook University
Industrial Technology Research Institute
chiueh@cs.stonybrook.edu

ABSTRACT

Credit and debit card payment processing systems are key elements in financial transactions. Negligence in securing these systems makes them vulnerable to hacking attacks, which may lead to significant monetary losses for both merchants and the financial organizations. To reduce this risk, mandatory security compliance regulations, such as the Payment Card Industry Data Security Standard (PCI DSS), were developed and adopted by the industry. A key pre-requisite of the PCI DSS compliance process is the ability to identify the components of the payment systems directly involved with the card data (i.e. process, transmit, or store). However, existing data flow tracking tools cannot fully automate the process of identifying system components that touch card data, because they either can not examine encrypted communications or they use an instrumentation-based approach and thus require a priori detailed knowledge of the payment card processing systems.

We describe the implementation and evaluation of a novel tool to identify the card data flow in commercial payment card processing systems running on virtualized servers. The tool performs real-time monitoring of network communications between virtual machines and inspects the memory of the communicating processes for unencrypted card data. Our implementation does not require instrumentation of application binaries and can accurately identify the system components involved in card data flow even when the communications among system components are encrypted. Effectiveness of this tool is demonstrated through its successful discovery of the card data flow of several open- and closed-source payment card processing applications.

1. INTRODUCTION

Among organizations increasingly targeted by ongoing cyber security attacks are retail businesses. These businesses make high-value targets for financially motivated cyber attackers because of the valuable credit and debit card data used in payment transactions. In the recent years, hackers have exploited weaknesses in payment card processing systems to steal sensitive customer card data [1].

To reduce security vulnerabilities in payment card processing systems, the Payment Card Industry Security Standards Council developed and released the Payment Card Industry Data Security Standard (PCI-DSS) [2]. All merchants that store, process, or transmit card data are required to comply with the PCI-DSS security requirements to ensure that not only the payment processing infrastructure, but the data it carries are better protected from unauthorized exposure. Noncompliant entities receive monthly fines and eventually may lose their ability to process card payments.

The key pre-requisite for PCI DSS compliance is the construction of the card data flow diagram for a payment processing network that accepts card charges and provides card processing service. Merchants must determine precisely how card data flow through their payment processing systems from their inception, what systems they traverse, and where they reside. A card data flow could start from a card swipe at a store, or a card number input by a user into an E-commerce web site, and consists of all intermediate stops in a merchant's IT network at which the card information is examined or processed. This discovery process and the resulting card data flow diagram help merchants understand which IT equipments in the organization interact with the card data so as to implement the security of these IT equipments according to the PCI DSS compliance requirements.

In practice, this pre-requisite poses a challenge to merchants. As the payment card processing infrastructure is implemented and later maintained, it often deviates from the originally documented design. Without consistent tracking and auditing of changes, such deviations in many cases remain undocumented. Today, no known tool exists that could automatically discover the card data flow of a distributed payment card processing system in heterogeneous computing environments. The only available solution to this problem today is manual card data flow reconstruction based on outputs from data loss prevention (DLP) tools and system design documents. DLP tools work by searching network packets and data stored on disk for clear text card numbers. Although highly effective when dealing with unencrypted data, the DLP tools are largely powerless when card data are encrypted in transit and on storage. Likewise, manual review of system design documents is an extremely labor-intensive and time-consuming effort. The required information is often difficult to extract because it is spread across a variety of IT elements and applications. Therefore building the card data flow for a given payment card processing infrastructure is considered a daunting task that at this point requires significant manual efforts.

The goal of this research is to develop an automated tool that is capable of building the card data flow in a distributed payment card processing system hosted on virtualized physical servers. We focus on virtualized servers because virtualization technology is quickly rising to predominate in merchants' data centers, and many payment card processing systems start to run inside virtual machines [3-5]. A key design decision of the tool we developed is to apply the virtual machine introspection (VMI) [6] technology to track card data flows.

Previous research efforts approached the automated data flow tracking problem from different angles, including a process-wide flow tracking, cross-process flow tracking, and cross-host flow tracking using fine-grained dynamic taint analysis (DTA). In DTA, data of interest are marked as tainted, and the taint propagation is monitored along with the data. The DTA data flow tracking mechanisms lead to increased level of detail, but either require a priori knowledge of the applications and hosts participating in information exchange so they can be properly instrumented or incur significant performance overheads that make such approaches unsuitable for interactive distributed network applications in production environments. Although our approach is more coarse-grained than the DTA methods and thus leads to a reduced level of detail in the produced data flow, it removes the need for application-specific instrumentations and the associated performance penalties.

In summary, this paper makes the following contributions:

- To the best of our knowledge, this is the first known tool to leverage VMI to automatically discover the card data flow of distributed applications running in virtualized environments. Because the VMI capability is already supported in modern hypervisors, such as Xen and VMware's ESX, our implementation does not require modifications to the hypervisor, VMs, guest OS, or payment card processing system components themselves.
- We have implemented a working prototype for the Xen hypervisor using VMI. We demonstrate the effectiveness of the tool by applying it to 3 commercial payment card processing systems and successfully building the card data flow path for each of them.
- We expect the availability of this tool could significantly decrease the efforts and costs in meeting the security regulations stipulated in the PCI DSS standard.

2. RELATED WORK

Several studies have explored the problem of data flow tracking in cross-host distributed systems. These can be roughly divided into dynamic binary instrumentation (DBI) and emulator-based implementations both using the DTA technique.

The data flow tracking tool described in [7] is built upon a DBI framework and is designed to track information flow between processes which may be located in different host systems. In this implementation, hosts and processes participating in the information flow are manually identified, and a DBI tool is then attached to each of the identified processes to track information flow within the process boundary. Additionally, a flow manager is placed in each participating host to relay taint information between interacting processes and to handle cross-host communications and data flow concatenations. In another related study [8], a single process DBI framework is extended to perform cross-process and cross-host transfer of taint information by intercepting and instrumenting the system calls used for cross-process as well as for cross-host communication. As these implementations require prior knowledge of the hosts and the processes involved in the information flow, these tools can not be utilized for data flow tracking where systems and processes participating in the data flow are unknown.

Several DTA studies explored the use of emulators to perform fine-grained taint propagation and tracking between processes and hosts in virtualized networks. In these implementations, the typical approach is to instrument hardware emulators, such as QEMU [9], with taint tracking instructions and monitor the taint propagation at the hardware level [10]. Taint tracking data structures are used to keep taint status flags of every byte in the system including physical memory, CPU registers, and device state. The emulator propagates taint flags whenever their corresponding values in hardware are involved in an operation.

In a related study, Data Flow Tomography [11] built on QEMU emulator implements fine-grained data flow analysis system to track and visualize data flow on a networked set of virtual machines each running on a separate physical host. The Data Flow Tomography method uses full instruction emulation and is inherently heavy weight both in memory and time. Hardware emulation is extremely slow and incurs significant performance overheads making this approach unsuitable for interactive network applications in production environment. To be a useful tool in the life cycle of a system, methods will be needed to speed up the analysis. While data flow tracking within a single machine is rarely problematic, the scalability of the approach as the number of nodes increases beyond two is certainly a question. This method also requires QEMU installation on every machine involved in the data flow.

Some research has been done to explore more efficient means for dynamic taint analysis. Zhang *et. al.* [12] implemented Neon, an extension of the [10] approach developed to prevent data leaks. Neon focuses on taint propagation across applications, systems, and networks. Neon implementation is based on the Xen hypervisor combined with demand emulation via QEMU, in which a running system dynamically switches between virtualized and emulated execution, and emulation is only used when tainted data is being processed by the CPU. This implementation leads to increased performance compared to using a processor emulator alone [11]. However, because propagating taint requires the invocation of QEMU, the Neon implementation incurs significant execution time overhead due to tag processing from the emulator and thus does not significantly improve performance.

Unlike previous approaches, we consider the most generic black-box approach that can be easily integrated into production environments, where no previous knowledge of the components participating in the data flow is provided, and only passive non-intrusive (i.e. require no modification of the monitored system) monitoring instruments with low performance impact are used.

This requirement was accomplished by leveraging the virtual machine introspection (VMI) technique. The term "virtual machine introspection" was introduced by Garfinkel and Rosenblum to describe a host-based intrusion detection system for virtual machines [6]. VMI refers to the method of monitoring and understanding the internal state of a running machine from the hypervisor by locating and examining internal data structures that in-guest APIs use. Using VMI, unobtrusive live system analysis may be performed on a target virtual machine without interfering with the target system's operation in any noticeable manner. Our tool utilized VMI technique to generate a list of active processes and open network sockets.

3. DESIGN AND IMPLEMENTATION

3.1 Payment Card Processing System

Out tool is designed for a payment card processing system consisting of multiple distributed application components all running on distinct VMs as separate processes and communicating with one another using synchronous requests. A payment request using a credit or debit card number is sent to the entry component in the system, e.g. a card swipe at a point-of-sale terminal at a store. Each application component forwards the request to the next component along the card processing path and blocks until the corresponding response is received. Once the payment card processing system verifies that an input request's card information is accurate and sufficient funds are available in the account, the request is granted permission to proceed with the purchase. Additional processing steps within the merchant's network may be triggered after a payment request is authorized, such as submission of payment data to storage, marketing data collection, payment reconciliation and settlement etc.

3.2 Assumptions

Two assumptions were made when developing the tool:

- Each card data handling component processes each request in a synchronous fashion, i.e., it reacts to an input request immediately and does not queue it for later processing.
- When applying the tool to a network to discover the card data flow, the network is in a "quiescent" state in the sense that only one test payment transaction is running through the payment system and a false positive caused by multiple concurrent requests is unlikely.

3.3 Requirements

The development of the tool is driven by the following requirements, which are derived from analyzing card data flows in real-world production environments:

- The tool does not require any modifications on the hypervisor, the guest OS or the application components of the target payment card processing system, and no additional software needs to be installed on the guest machines on which the payment system runs.
- The tool does not make any assumptions on the internal operations of the target payment application system being tracked other than the following: (1) The target application runs on a virtualized environment, and (2) Credit and debit card numbers are transiently stored in memory in a particular form.
- The tool imposes minimal performance overhead and operates seamlessly in the background with the target payment card processing system running at full steam.

3.4 System Overview

To identify the trajectory of the card data flow, a payment request is sent to the entry point of a payment card processing system, and the tool is employed to determine the set of VMs and the corresponding processes exchanging network packets as a result of this request (Figure 1).

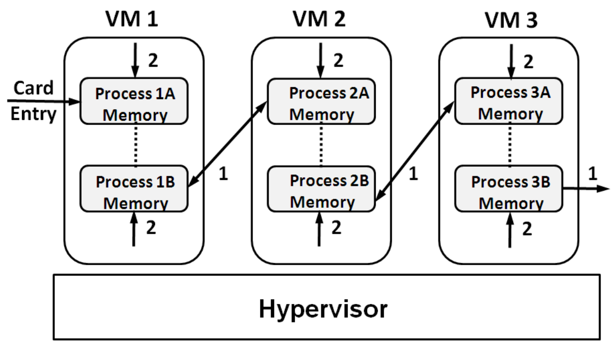


Figure 1. (1) Inter-VM network communications are tracked by the tool, and (2) the memory of the interacting processes is inspected for card data.

Because network communications among payment system processes may be encrypted, it is not always possible to detect card data from intercepted network packets. Therefore, the tool searches the memory spaces of the communicating processes for the card data as they travel from the entry-point process to other card data handling processes along the way. Even though card data may be encrypted during their IPC transmissions, they are decrypted and operated on during their processing, and therefore the clear text version of card data can be traced in the interacting processes' memory. Once the processes whose memory contains card data are found the machines involved in the card data flow are readily identified.

The card data flow trajectories from multiple virtual machines spread over several physical hosts can be further concatenated to determine how card data flow among networked hosts within the organization (Figure 2).

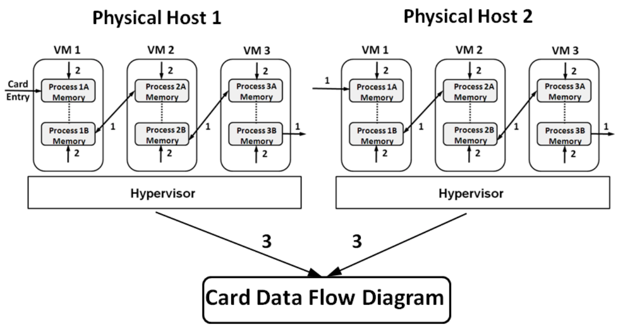


Figure 2. Card data flow concatenation from multiple physical hosts.

3.5 Main Components

We implemented the card data flow tracking tool for the Xen hypervisor [13] and fully-virtualized (HVM) Windows-based VMs (payment card processing systems predominantly run Windows OS).

Xen supports two types of virtual machines: unprivileged domains, called DomU domains, and a single privileged Linux-based domain, called Dom0. In our implementation, we deploy the tool in Dom0 and run the components of the payment card processing system in DomUs. The prototype implementation consists of two agents: the network agent and the introspection agent both running in Dom0 (Figure 3). The algorithmic outline involving these components comprises the following high-level steps:

(1) The network agent recursively traces inter-VM TCP connections starting from the entry-point VM that receives the test input request.
(2) The introspection agent searches the memory space of communicating processes bound to the intercepted TCP connections for the card number used as a test input at the entry-point VM.
(3) The card data flow path is reconstructed based on the results from (1) and (2).

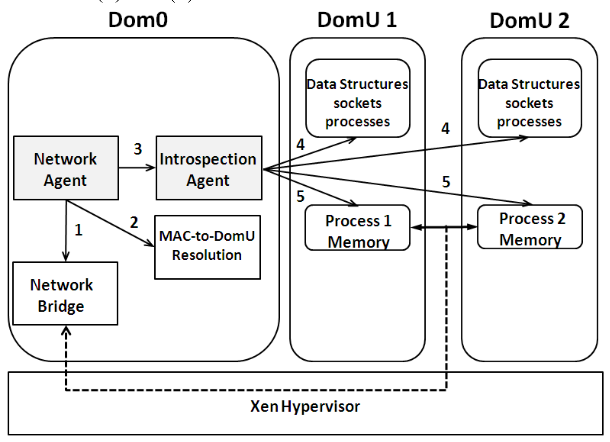


Figure 3. (1) Network connections are intercepted; (2) MAC addresses are resolved to DomU IDs; (3-4) Introspection agent is invoked to determine the processes participating in the network connections; (5) Process memory is searched for card data.

3.5.1 Recursive Tracing of Inter-VM Communications

The network agent is a user-space program running in Dom0 that tracks inter-VM network communications starting from the entry-point DomU VM that receives the test input request. It intercepts network packets and examines their headers by hooking into Xen's network bridge.

The network agent makes use of the packet filtering tool *ebtables* to intercept all packets sent to or from DomUs. Ebtables is an open source utility that filters packets at an Ethernet bridge [14]. As of the 2.6 Linux kernel, the ability to perform bridge mode filtering using ebtables is natively included in the kernel and supported by default. Through command line arguments, ebtables is instructed to pass intercepted packets to the network agent using netlink sockets. When the network agent receives a packet from ebtables, it parses the packet to extract its source and destination MAC addresses and port numbers (src MAC, src port, dst MAC, dst port) from the packet header. The src and dst MACs are then resolved to the DomU IDs using XenStore. In Xen, XenStore stores information about each DomU during its execution including the domain IDs and the corresponding MAC addresses. The network agent initiates a memory search request by sending the resulting (src DomU ID, src port, dst DomU ID, dst port) to the introspection agent.

The network agent invokes introspection requests in a multi-threaded fashion and never blocks on these requests allowing the introspection agent to perform the VM analysis in parallel using separate threads.

The network agent maintains a table of all the (src MAC, src port, dst MAC, dst port) connections being currently analyzed by the introspection agent to avoid issuing redundant requests while the request processing is in progress. Upon completion of an introspection request, the corresponding connection record is removed from this connection table.

3.5.2 Search of the Process Memory

The introspection agent is a user-space program running in Dom0 that receives the (src DomU ID, src port, dst DomU ID, dst port) information for each active network connection from the network agent, identifies the processes associated with each network connection, and searches the memory of the identified processes for the card number used in the test input.

The main algorithm of the introspection agent consists of the following three steps.

(1) The agent maps the physical memory pages of the source and destination DomUs to Dom0, so that it can analyze their contents.

(2) The agent parses the mapped pages to extract open sockets and identify the processes bound to the source and destination sockets of an inter-VM connection.

(3) The agent identifies the portions of the mapped physical memory space that belongs to the identified processes, so that it can focus on those portions only, and searches these memory portions for the test card number used in the test transaction.

The introspection agent applies the above algorithm starting with the entry-point process and continues recursively on each intercepted network connection.

3.5.2.1 Accessing DomU Memory

By design, Dom0 is granted complete access to the entire state of the guest operating systems running in DomUs and can determine execution properties of DomUs by monitoring their run-time state through direct memory inspection. Xen offers low-level APIs to allow Dom0 to map arbitrary physical memory pages of a DomU to its memory space. These APIs operate on DomU IDs obtained from the network agent. XenAccess is a Dom0 user-space VMI library developed for Xen that is built upon the low-level APIs provided by Xen [15]. We leverage the PyXaFS file system, which is part of the XenAccess tool suite, to map physical memory pages of DomU's kernel inside Dom0. PyXaFS exposes the memory of a DomU as a regular file on the host and allows the introspection agent to read a live DomU's memory as if it were a normal file.

3.5.2.2 Parsing Sockets and Processes Structures

Given a DomU's physical memory space, the introspection agent bridges the so-called semantic gap [16-18] between low-level memory pages and high-level kernel data structures, such as network connections, sockets, and process list. Because in-guest APIs are not accessible from Dom0, it is non-trivial to reverse-engineer guest OS-specific constructs, especially for a closed-source operating system such as Windows OS, which is the target of this project. We leverage Volatility Framework [19] to solve the problem of network socket and process identification in a live VM's physical memory pages.

Volatility Framework is an open source Python-based framework that was specifically designed to assist forensic investigators with the examination of volatile memory. The Volatility Framework currently supports Windows XP, Windows 2003 Server, Windows Vista, Windows 2008 Server, and Windows 7. The extraction techniques utilized by Volatility include such capabilities as obtaining the list of running processes, open network sockets and connections, ability to dump a process' addressable memory, etc.

Applying the Volatility Framework to a DomU's mapped physical memory space allows the tool to extract the process IDs bound to the network sockets matching port numbers intercepted by the network agent.

Although our implementation is based on the Volatility framework alone, numerous forensic tools exist that can handle various OS versions and flavors and can substitute for Volatility.

3.5.2.3 Searching the Process Memory
Given a process ID obtained in the above step, the introspection agent finds the physical memory pages owned by the process and searches only that process's physical memory pages for the test card number using the following patterns. Payment card numbers are sequences of 13 to 16 digits. The card issuer is identified by a few digits at the start of these sequences. For instance, Visa card numbers have a length of 16 and a prefix value of 4. MasterCard numbers have a length of 16 and a prefix value of 51-55. Discover card numbers have a length of 16 and a prefix value of 6011. Finally, American Express numbers have a length of 15 and a prefix value of 34 or 37. Therefore, finding these card numbers in memory can be accomplished by searching for ASCII strings that match the following regular expression: ((4\d{3})|(5[1-5]\d{2})|(6011))-?\d{4}-?\d{4}-?\d{4}|3[4,7]\d{13}.

However, sequences of 13 to 16 digits with proper prefix values are not always card numbers. Each potential card number obtained by the above search procedure has to be further verified using the Luhn algorithm [20], which is a simple checksum formula that is commonly used to validate the integrity of a wide variety of identification numbers.

3.5.3 Card Data Flow Reconstruction
To build the card data flow, the processes whose memory contains the test card data and the communication connectivity among them are combined into a graph. When two processes of a payment card processing system communicate, there are three possible state combinations after searching their memory pages, as shown in Figure 4(A): (1) The test card data found in the memory of both processes, (2-3) the test card data found in the memory of either process but not both, and (4) the test card data is found in the memory of neither process.

Similarly, when the introspection agent scans a process's memory in a VM that serves as a card receiver and as a card sender, there are three possible state combinations, as shown in Figure 4(B).

When the introspection agent could not find the test card number in a process's memory, there are three possible explanations. First, the process does not receive the test card data at all. Second, the process receives an encrypted version of the test card data, but does not decrypt it. Third, the process receives the test card data either in clear text or in encrypted form, but the introspection agent scans the process at an inopportune time, e.g. before the decryption of an encrypted card number or after the clear text card number is overwritten.

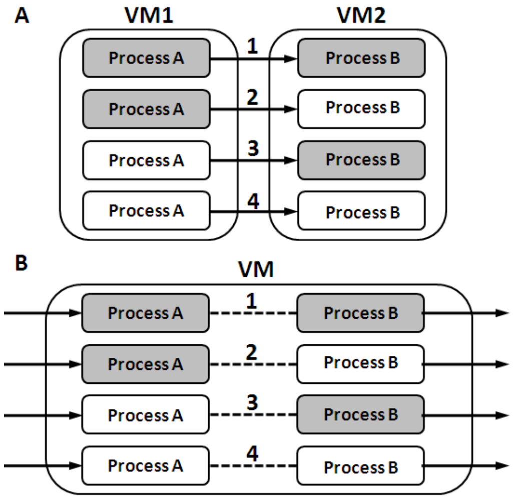


Figure 4. (A) 4 possible states of two inter-VM communicating processes (grey rectangle - the card number found in process memory, white rectangle - no card number found in process memory. The arrow indicates the direction of connection initiation, not traffic flow); (B) 4 possible states of processes within a VM at packet receiving time and at packet sending (the same process may serve as the receiving and sending process).

3.5.4 Optimization
To increase the probability of card detection, the tool scans each communicating process multiple times. The first scan examines every memory page in the process. If the card number is not found in the first scan, the tool re-scans the memory. Each subsequent scan only inspects those memory pages that are modified since the last scan. We exploited Xen's dirty page tracking capability originally designed for live migration to identify modified pages between consecutive scans. This incremental scanning approach drastically decreases the card number search overhead in subsequent scans. If no card number is found after a specified number of scans of a given process, the introspection agent assumes the process is not in the card data flow.

Just because no card number is found in a process does not mean that the process cannot be part of a card data flow. For example, the process can receive an encrypted card number and pass it on to the next process without decrypting it. Therefore, the introspection agent has to scan all communicating processes regardless of whether the sending process contains the test card number.

4. EVALUATION
In this section, we describe experiments demonstrating distributed card data flow tracking using our tool. We tested the tool on 3 payment card processing systems: two e-commerce shopping carts and a point-of-sale system (Table 1).

Table 1. Evaluation suites and testing results.

Payment Card Processing System Name	Able Commerce System	osCommerce System	CreditLine System
Software Description	Commercial shopping cart system used by more than 10,000 stores worldwide [21]	Free e-commerce and online store-management software program [22] used by over 12,000 online shops worldwide	Payment processing client-server application designed as point-of-sale system [23]
Software Language & Platform	ASP.NET/ MSSQL	PHP/MySQL	Binary Windows executable
DomU Client Component	Internet Explorer browser	Internet Explorer browser	Client application
DomU Server Component	IIS 5.1 web server with .NET framework v3.5	IIS 5.1 web server running PHP v5.3.3	Server application
DomU DB Component	MSSQL Express Server 2005	MySQL 5.1.52	N/A
Encryption in Transit	SSL	SSL	N/A
Results	The test card number was found in memory of Client and Server DomUs.	The test card number was found in memory of Client, Server, and DB DomUs.	The test card number was found in memory of Client and Server DomUs.
Average Time to Identify the Flow, sec	9	7	8

4.1 Card Data Flow Tracking Across Multiple VMs Hosted on the Same Physical Host

Our testbed consisted of a virtualized server that used Xen version 3.3 as the hypervisor and Ubuntu 9.04 (Linux kernel 2.6.26) as the kernel for Dom0. The tool was installed in the Dom0. In addition, the virtualized server hosted 3 DomU domains (DomU Client, DomU Server, DomU DB) running Windows XP. The payment card processing systems were installed in these 3 domains as outlined in Table 1 and were running simultaneously to mimic the real-world production environments with multiple services running on the communicating hosts.

When conducting our experiment, we selected several items for purchase and submitted credit card information at checkout. Following the payment card processing requests, the tool determined the set of machines exchanging packets, identified the processes involved in these communications, and inspected the processes' memory for the card number used in the transaction, while allowing the applications to run throughout the analysis. The testing results are presented in Figure 5.

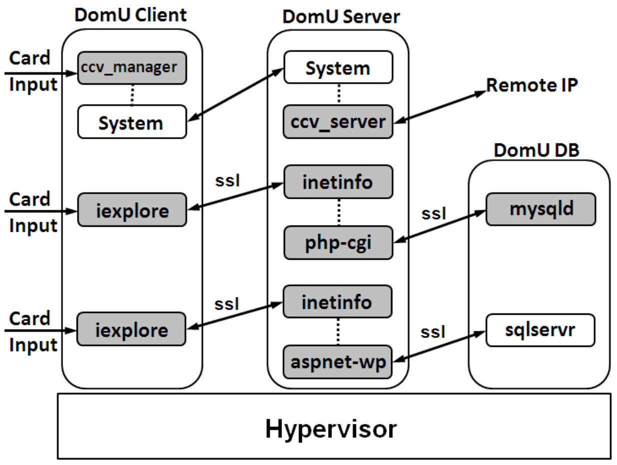


Figure 5. Processes involved in card data flow (CreditLine flow at the top, osCommerce flow in the middle, and AbleCommerce flow at the bottom).

Additionally, we also captured network packets exchanged between machines to determine if an accurate card data flow could be built by only inspecting the contents of the sniffed packets without applying the tool. As expected, we could not detect the test card number in the sniffed packets due to the SSL encryption configured on these communications (Figure 6-8).

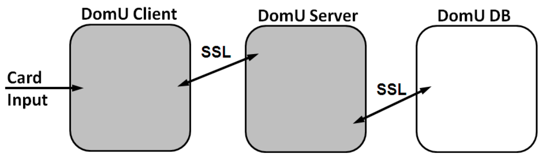
In some cases, the tool was also able to identify other card related information including the card expiration date, CVV number, and the cardholder's name within the same memory segment as the corresponding card number as shown in Figure 9.

When running our tests, we observed the timings and the portions of memory from which card data were extracted and classified the card data extraction instances into four categories:

(1) Transient/Stack: The card data are uncovered from a stack region while the associated transaction is being processed.
(2) Persistent/Stack: The card data are uncovered from a stack region after the associated transaction is completed.
(3) Transient/Heap: The card data are uncovered from a heap region while the associated transaction is being processed.
(4) Persistent/Heap: The card data are uncovered from a heap region after the associated transaction is completed.

The successful card data extractions the introspection agent is able to perform against the test payment card processing systems fall into category (1), (3) and (4). Category (2) is rare because memory words allocated on the stack are automatically freed and possibly overwritten when they are no longer needed. In contrast, memory words allocated from the global heap have a much longer life time, because application programs need to explicitly free them when they are no longer needed, but application programs rarely do so. As a result, card data stored on the heap exist for at least the duration of the associated transaction, which typically takes up a few seconds to complete, and in many cases continue to exist even after the associated transaction is completed.

(a)



(b)

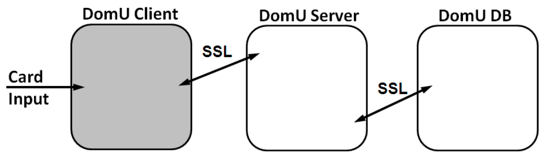
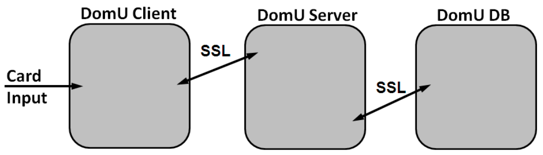


Figure 6. AbleCommerce Card Data Flow (machines found to participate in the card data flow are shown in grey) (a) using our tool; (b) using a packet sniffer.

(a)



(b)

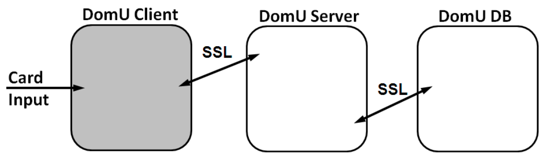
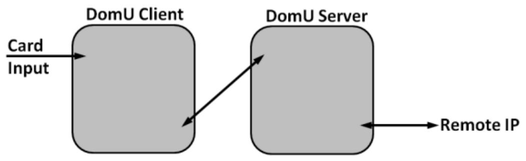


Figure 7. osCommerce Card Data Flow (machines found to participate in the card data flow are shown in grey) (a) using our tool; (b) using a packet sniffer.

(a)



(b)

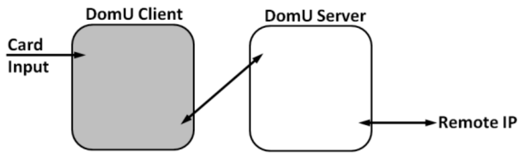
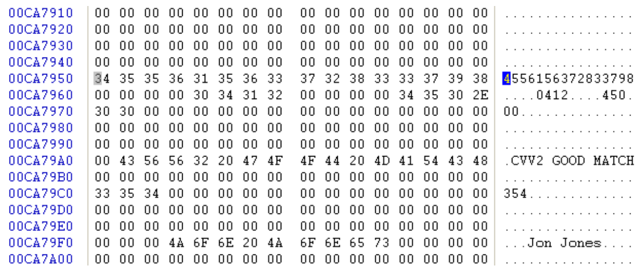


Figure 8. CreditLine Card Data Flow (machines found to participate in the card data flow are shown in grey) (a) using our tool; (b) using a packet sniffer.



```
00CA7910  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
00CA7920  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
00CA7930  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
00CA7940  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
00CA7950  34 35 35 36 31 35 36 33  37 32 38 33 33 37 39 38  4556156372833798
00CA7960  00 00 00 00 30 34 31 32  00 00 00 00 34 35 30 2E  ....0412....450.
00CA7970  30 30 00 00 00 00 00 00  00 00 00 00 00 00 00 00  00..............
00CA7980  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
00CA7990  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
00CA79A0  00 43 56 56 32 20 47 4F  4F 44 20 4D 41 54 43 48  .CVV2 GOOD MATCH
00CA79B0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
00CA79C0  33 35 34 00 00 00 00 00  00 00 00 00 00 00 00 00  354.............
00CA79D0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
00CA79E0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
00CA79F0  00 00 00 4A 6F 6E 20 4A  6F 6E 65 73 00 00 00 00  ...Jon Jones....
00CA7A00  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
```

Figure 9. Detailed information uncovered about a test card, including the card number (4556156372833798), the card expiration date (0412), the CVV number (354), and the cardholder's name (Jon Jones) were identified within the process memory.

4.2 Card Data Flow Tracking Across Multiple VMs Hosted on Multiple Physical Hosts

All communications in the first experiment occur between VMs running on top of the same hypervisor, while in the real world the processes in a payment card processing system are more likely to reside in multiple VMs spread over multiple physical hosts. In the following experiment, we demonstrate the capability of our tool to work equally effective in a multi-physical-host setting.

Our testbed consisted of three virtualized servers that used Xen version 3.3 as the hypervisor and Ubuntu 9.04 (Linux kernel 2.6.26) as the kernel for Dom0. The tool was installed in the Dom0 on each physical host. Each physical host was running one DomU domain. The first virtualized server hosted DomU Client, the second virtualized server hosted DomU Server, and the third virtualized server hosted DomU DB all running Windows XP. The payment card processing systems were installed in these 3 domains as outlined in Table 1 and were running simultaneously to mimic the real-world production environment with multiple services running on the communicating hosts.

When conducting our experiment, we selected several items for purchase and submitted credit card information at checkout. Following the payment card processing requests, the tool determined the set of machines exchanging packets, identified the processes involved in these communications, and inspected the processes' memory for the card number used in the transaction, while allowing the applications to run throughout the analysis. The card data flow trajectories from the 3 virtual machines spread over 3 physical hosts were then concatenated to build the card data flow. The testing results are presented in Figure 10.

5. LIMITATIONS AND FUTURE WORK

In this paper, we presented the design and implementation of a payment card data flow tracking tool, which is capable of reconstructing the card data flow in a payment card processing system running on a virtualized environment. We discuss below some of the limitations of this tool and opportunities for further work.

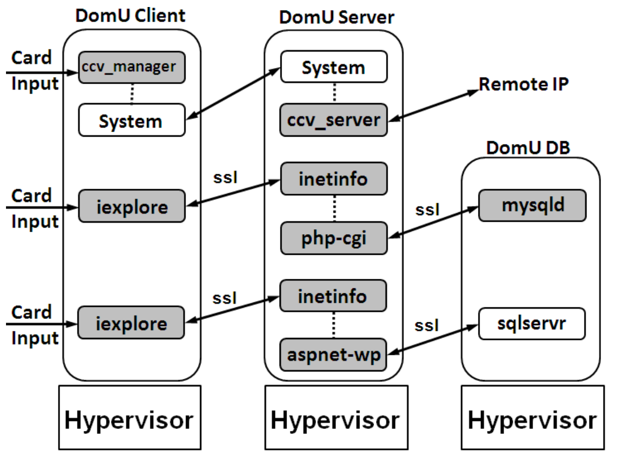


Figure 10. Card data flow across multiple VMs hosted on multiple physical hosts.

The test environments used to date have been useful in demonstrating the effectiveness of the tool but they are rather simplistic and do not display many of the characteristics of large-scale deployments. Although we tested three different settings, they all involved just four processes distributed across three VMs interacting in almost identical fashion. Unlike the simple test scenarios described in this work where the number of factors influencing the correctness of the data flow reconstruction is minimal, the task of the card flow identification becomes increasingly more complex in real-world production setups. For instance, if two VMs communicate for reasons not related to payment data flow, such as periodic updates, heartbeats, replications, backups, other services running on the communicating hosts, and so on, then these connections may be mixed up with those for card data processing. These additional communications could significantly increase the workload of the card data flow tracking tool. Moreover, network delays may also critically affect the ability to track a card number within a process. More complex environments may introduce a race condition that may affect the effectiveness of the system: the target process (which may be running on a different physical machine) may have completed its processing (all of it, or just the part that involves the card data in its unencrypted form) before the introspection agent on that physical machine manages to analyze the process' memory. Additionally, we have also assumed that the system is in a “quiescent” state where only one simulated transaction takes place at a time. This assumption is quite restricting from a practical point of view, given that in a production setting it would be quite difficult to ensure that there are no other ongoing transactions. To take into account the above factors, the next stage of our work is to conduct additional evaluations of our tool on testbeds that mimic production environments to identify possible limitations of the tool’s current design or implementation.

In our implementation, we monitor TCP connections to track card data flow across multiple VMs. Specifically, the tool initiates a search of the memory of a VM only when it is involved in a newly established TCP connection. In the future, we plan to support persistent TCP connections, which may stay open for a long time and service multiple transactions. Also, our coarse-grained data flow tracking mechanism does not currently handle data flow tracking of cross-process communications within the same VM. More research is required to determine the extent to which data flow tracking can be implemented via cross-process intra-host TCP connections, pipes, and shared memory. This is an important direction of future research.

Finally, it is possible that a card number can be handled by processes in an encrypted form and is never decrypted during its processing, as revealed by some of our experiments. This issue will affect the accuracy of the derived data flow diagram our tool produces.

In spite of the above limitations, we consider our work as a proof of concept, aiming to motivate further research in this space. We plan to address these limitations in the next version of the tool.

6. SUMMARY AND CONCLUSIONS

This paper presents a novel tool that can automatically track card data flow of payment card processing applications running in a virtualized environment and identify the system components involved in card data processing. The primary use of this tool is to ensure compliance with Payment Card Industry Data Security Standard (PCI DSS) that has been widely adopted by commercial and financial institutions. The key features of this tool include: 1) the ability to discover the card data flow of a distributed payment card processing system; 2) independence of applications and platforms; and 3) the ability to deal with communication protocols that encrypt messages. We have demonstrated this tool’s effectiveness by testing it with three different commercial applications, and the tool successfully identified the correct card data flow for each tested application.

The main difference between our tool and other tools with a similar goal is that it applies virtual machine state inspection, and therefore does not rely on machine or application instrumentation and imposes minimal performance overhead when dealing with multiple machines. The tool’s major limitations include lack of inter-process data flow tracking capability within the same machine and inability to follow card data flows that never decrypt card data during their processing. Despite these limitations, we believe the conceptual framework devised in this work could be applied to designing similar tools for real-world payment card processing applications running on real-world computing environments.

7. REFERENCES

[1] Privacy Rights Clearinghouse, Chronology of Data Breaches. http://www.privacyrights.org/ar/ChronDataBreaches.htm.

[2] PCI Security Standards Council. https://www.pcisecuritystandards.org/.

[3] Bringing Virtualization and Thin Computing Technology to POS. http://www.pippard.com/pdf/virtualized_pos_whitepaper.pdf

[4] Restaurant Chain Upgrades Systems and Cuts 2,000 Servers Using Virtual Machines. http://download.microsoft.com/documents/customerevidence/7146_jack__in_the_box_cs.doc.

[5] Micros Systems, Inc. Announces Deployment of Micros 9700 Hms at M Resort Spa Casino in Las Vegas. http://www.micros.com/NR/rdonlyres/3E357BE8-70DB-468D-B9AB-68F0E784527F/2296/MResort.pdf.

[6] Garfinkel, T. and Rosenblum, M. A Virtual Machine Introspection Based Architecture for Intrusion Detection. In *Proceedings of the 10th Annual Symposium on Network and Distributed System Security*, 2003.

[7] Kim, H.C., Keromytis, A.D., Covington, M., and Sahita, R. Capturing Information Flow with Concatenated Dynamic Taint Analysis. In *International Conference on Availability, Reliability and Security*, 2009.

[8] Zavou, A., Portokalidis, G., and Keromytis, A.D. Taint-Exchange: A Generic System for Cross-Process and Cross-Host Taint Tracking. *The 6th International Workshop on Security*, 2011.

[9] Bellard, F. Qemu, a Fast and Portable Dynamic Translator. In *Proceedings of the 2005 USENIX Annual Technical Conference*, 2005.

[10] Ho, A., Fetterman, M., Clark, C., Warfield, A., and S Hand. Practical Taint-Based Protection Using Demand Emulation. In *Proceedings of the 1st ACM SIGOPS/EuroSys European Conference on Computer Systems*, 2006.

[11] Mazloom, B., Mysore, S., Agrawal, B., and Sherwood, T. Understanding and Visualizing Full Systems with Data Flow Tomography. In *ASPLOS'08*, 2008.

[12] Zhang, Q., McCullough, J., Ma, J., Schear, N., Vrable, M., Vahdat, A., Snoeren, A. C., Voelker, G. M., and Savage, S. Neon: System Support for Derived Data Management. In *Proceedings of the 6th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments*, 2010.

[13] What Is Xen? http://www.xen.org/.

[14] Ebtables. http://ebtables.sourceforge.net/.

[15] Xenaccess Documentation. http://doc.xenaccess.org/.

[16] Payne, B. D., Carbone, M., Sharif, M., and Lee, W. Lares: An Architecture for Secure Active Monitoring Using Virtualization. In *Proceedings of the 2008 IEEE Symposium on Security and Privacy*, 2008.

[17] Jiang, X., Wang, A., and Xu, D. Stealthy Malware Detection through Vmm-Based "Out-of-the-Box" Semantic View Reconstruction. In *Proceedings of the 14th ACM conference on Computer and communications security*, 2007.

[18] Jones, S. T., Arpaci-Dusseau, A. C., and Arpaci-Dusseau, R. H. Antfarm: Tracking Processes in a Virtual Machine Environment. In *Proceedings of the 2006 USENIX Annual Technical Conference*, 2006.

[19] The Volatility Framework: Volatile Memory Artifacts Extraction Utility Framework. https://www.volatilesystems.com/default/volatility.

[20] Luhn, H.P. Computer for Verifying Numbers, USA, U. S. P. Office, 1954.

[21] Ablecommerce: Features Clients. http://www.ablecommerce.com/Featured-Clients-C49.aspx.

[22] Oscommerce: Open Source E-Commerce Solutions. http://www.oscommerce.com/.

[23] Payment Processing Software. http://www.911software.com/.

An Empirical Study of Visual Security Cues to Prevent the SSLstripping Attack

Dongwan Shin
Computer Science and Engineering
New Mexico Tech
Socorro, NM 87801, USA
doshin@nmt.edu

Rodrigo Lopes
Computer Science and Engineering
New Mexico Tech
Socorro, NM 87801, USA
rodrigo@nmt.edu

ABSTRACT

One of the latest attacks on secure socket layer (SSL), called the *SSLstripping* attack, was reported at the Blackhat conference in 2009. As a type of man-in-the-middle (MITM) attack, it has the potential to affect tens of millions of users of popular online social networking and financial websites protected by SSL. Interestingly, the attack exploits users' browsing habits, rather than a technical flaw in the protocol, to defeat the SSL security. In this paper we present a novel approach to addressing this attack by using visually augmented security. Specifically, motivated by typical traffic lights, we introduce a set of visual cues aimed at thwarting the attack. The visual cues, called security status light (SSLight), can be used to help users make better, more informed decisions when their sensitive information need to be submitted to the websites. A user study was conducted to investigate the effectiveness of our scheme, and its results show that our approach is more promising than the traditional pop-up method adopted by major web browsers.

1. INTRODUCTION

There has been growing interest in usable security over the last decade, and numerous studies have been conducted to explore the usability issue of various computer security mechanisms [8, 19, 21, 22]. However, bridging the gap between security and usability is a challenging task. At one extreme, we can easily find security technologies, for example, one-time pads (OTPs), that render many sophisticated attacks obsolete but are not widely adopted due to their usability issues. At the other extreme, we can find security solutions, for instance, low entropy passwords, that are very usable but fail to withstand non-trivial attacks.

One of the common mistakes made when designing security systems is that security is assumed to be both important and engaging for users. However, this is clearly not a correct assumption since security is almost never the users' primary goal [22]. The security ramification of this disengagement is the constant surfacing of new attacks that exploit not only technological flaws, but also usability flaws. One such example is the *SSLstripping* attack introduced at the Blackhat conference in 2009 [11]. It attacks secure socket layer (SSL), which is the most widely used security mechanism that enables secure communication establishment between two parties over the Internet. As a type of man-in-the-middle (MITM) attack, the attack could affect tens of millions of online users of popular SSL-protected websites such as Facebook.com. More importantly, the attack exploits users' browsing habits, rather than a technical flaw in the protocol, to effectively defeat the SSL security.

The underlying cause of this type of semantic attack, which targets the way we, as humans, assign meaning to content [14], is a semantic barrier between the user's mental model and the system's actual processing model [21]. Since web interactions between a user and a remote web server are usually made through the use of a web browser, the user derives her mental model of the interactions from the presentation of the interactions - the way they appear on the web browser screen. On the other hand, the system's processing model is derived from the messages exchanged between the web browser and the web server, without a web browser being aware of the user's intention. Hence, the user, who usually does not look at the hidden HTML codes, is not able to compare her mental model ("*I use a secure channel to submit my login credential*") with the system's processing model ("*A secure channel is not used due to an attack*") under the SSLstripping attack. The semantic barrier further results in the lack of trust between the user and her agent.

In this paper we present a novel approach to addressing this type of semantic attack, especially, the *SSLstripping* attack, by using visually augmented security. Specifically, motivated by the red-yellow-green colored traffic light metaphor, we introduce a set of visual cues aimed at thwarting the attack. The visual cues, called security status light (SSLight), can be used to boost the user's trust against her browser by trying to overcome the semantic barrier between the user's mental model and the system's processing model. Hence, users can make better, more informed decisions when their sensitive information needs to be submitted to a remote website. An experimentation involving a user study was designed and conducted to investigate the effectiveness of our scheme, and its result shows our approach is more effective than the traditional pop-up method adopted by major web browsers.

This paper is organized as follows: Section 2 discusses the SSLstripping attack and its countermeasures in detail. Section 3 presents our approach to tackle the attack, with an

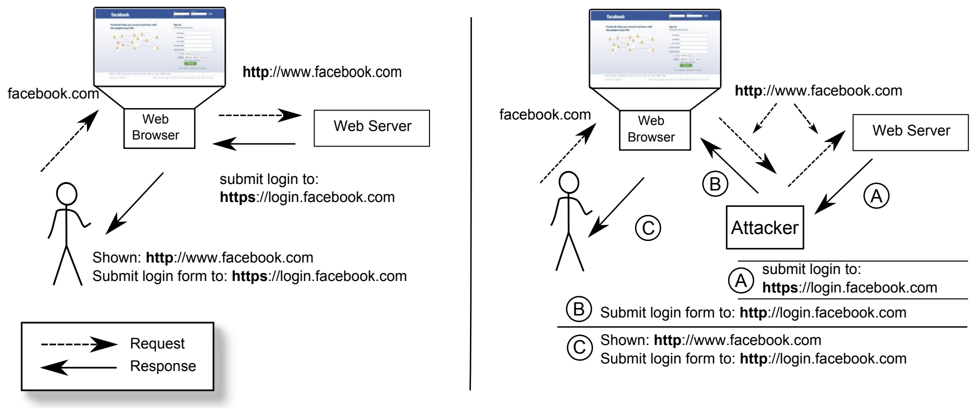

facebook.com
http://www.facebook.com
Web Browser
Web Server
submit login to: https://login.facebook.com
Shown: http://www.facebook.com
Submit login form to: https://login.facebook.com
Request
Response
facebook.com
http://www.facebook.com
Web Browser
Web Server
B
A
C
Attacker
A submit login to: https://login.facebook.com
B Submit login form to: http://login.facebook.com
C Shown: http://www.facebook.com
Submit login form to: http://login.facebook.com

Figure 1: User interaction with Facebook.com in a normal situation (left) and under SSLstripping attack (right)

introduction of SSLight. Section 4 discusses the methodology and design of a user study developed to test the effectiveness of our approach. Section 5 and 6 present the results of the user study. Section 7 concludes this paper with our future research direction.

2. BACKGROUND AND RELATED WORK

Since SSL was first introduced by Netscape in 1994, it has become a de facto standard for establishing a secure connection between two parties over the Internet. It also laid the foundation for developing a new standard called transport layer security (TLS) [4, 5]. Nowadays, many websites rely on HTTPS, HTTP served over a SSL/TLS channel, to provide users a secure transaction over the Internet. It is interesting, however, to see that nearly 45% of the most *popular* websites still do not use HTTPS, not even for login purposes, as shown in a recent study [16].

2.1 SSLstripping Attack

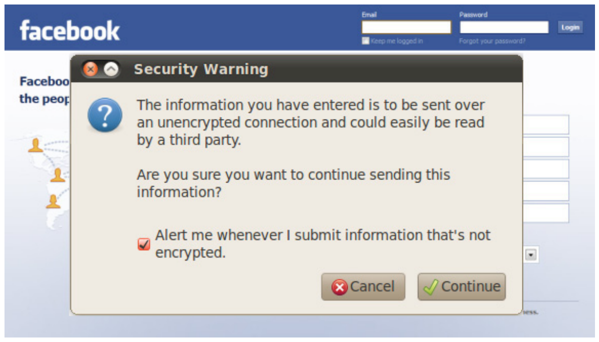
The man in the middle (MITM) attack has been a constant threat in web security. A MITM attack is achieved when a malicious user positions himself between two other users that intend to engage in a legitimate communication. Effectively, in a MITM attack, the attacker impersonates each of the victims to the other. A variation of MITM attack called the SSLstripping attack was proposed at the Blackhat conference in 2009 [11]. The attack takes advantage of the simple concept and observation that most users do not explicitly type the safe address of a web page (https), but rather rely either on the browser or the target page to redirect them to a secure location. This opens the opportunity to strip users' sessions of its security, while giving the user the illusion of privacy. More specifically, the attack works as follows:

- The attacker is on the same local network as the victim
- The attacker uses the ARP protocol to convince the victim that the attacker's machine is the victim's gateway
- The victim, unknowingly, is now sending all her HTTP requests to the attacker
- The attacker will look at all HTTP requests and do either of the following:
  - Case 1: If the response is a redirection to an HTTPS address, the attacker will establish the secure connection to the address and provide the decrypted content to the victim, also stripping 's' from "HTTPS" in any forms and links
  - Case 2: If the response is a mixed content page, which is an HTTP page with forms with "HTTPS" actions, the attacker will simply strip 's' from "HTTPS"
- The victim logs in to a service
- The attacker will receive the victim's login request and:
  - Register the victim's credentials
  - Login in place of the victim with the web server
  - Provide the victim with the received responses, after decryption
- The victim, unaware of the attack, experiences the same as she would experience if the connection was secure. Her browser issues no warnings, since no SSL certificates are being used on the victim's machine.

As shown by Marlinspike at the conference and verified later at our lab, this attack is really easy to launch and extremely effective: it would be detected only by very careful and technically savvy users by either noticing the insecure connection on a normally secure website (Case 1) or by looking at the rendered web page's source code and verifying that the login form does not have a secure address as its target (Case 2). Figure 1 illustrates this attack by using Facebook.com.

2.2 SSLstripping Countermeasures

Most of the current web browsers have little or no support for security indicators when users submit their sensitive information over an insecure communication channel. Hence, it is hard for users to have any suspicion that something should be wrong when they are under the SSLstripping attack. Both Mozilla Firefox and Internet Explorer support



The information you have entered is to be sent over an unencrypted connection and could easily be read by a third party.

Are you sure you want to continue sending this information?

Alert me whenever I submit information that's not encrypted.

Figure 2: The classic Pop Up Window warning present in Mozilla Firefox

a warning dialog when users submit their data over an insecure channel, as shown in Figure 2. However, this is an opt-in feature which users need to explicitly choose to see the warning. Other browsers, such as Google Chrome, do not have even the opt-in warning dialog. However, a warning feature could be supported by installing browser extensions in Google Chrome.

Jackson and Barth suggested a solution where the user's web browser would force a secure connection with websites that have previously deployed a special cookie [9]. This solution will not be able to avoid the SSLstripping attack in general, since most websites do not have the **SSL-only** requirement. Nikiforakis et al. proposed an approach to leveraging a client side HTTP proxy, which, relying on browser history, would compare the current request with previous requests made to the same website in [12]. Although this is an interesting approach, it fails to address several issues. First of all, it requires a browser history. This automatically excludes all users that do not save browsing history. Second, it assumes that the attack is never a zero day attack. This excludes all users that were already attacked. The assumption that the attack will only occur after a significant browser history is established, seems to be a very weak assumption.

2.3 HTTPS Visual Security Cues

Whalen and Inkpen conducted a study on the effectiveness of different security indicators in [18]. Their study used an eye-tracking device to match users' claims to their actual actions, more specifically where they focused their attention. The study had several interesting results. First of all, on top of the previously identified security indicators such as a padlock and https in an address bar, more security indicators were identified, specially concerning a web site reputation, either by general user rating or by the level of usage among a user's circle of friends. Furthermore, the study was run in two phases, the first where users were not prompted to pay special attention to security aspects, while in the second they were. They showed that the resulting differences between the two stages of the study were very significant: users would normally not pay attention to the security indicators. Overall, the lock displayed on a browser in the event of a secure connection being established was the preferred security indicator. However, it is important to notice that many browsers allow a page to display a small icon on the address bar, which can be made to look like a lock regardless of a secure connection being established or not. Additionally, it was found that some users were confused with other small icons for the lock icon.

In their study, Schechter et al. found out that removing the SSL security indicators would not deter users from logging in to different websites with their credentials [13]. But this phenomenon could be explained partially by the existence of different security indicators, not directly related to SSL, as identified by Whalen and Inkpen [18]. On the other hand, the effect of introducing extended certificates, aiming to provide even more information about service providers to users, was studied in [10, 15]. However, it was found that users also ignored the information on these certificates.

Usable security issues were not limited to the SSL usage for secure web browsing. There have been several studies on the usability of PGP encryption and effectiveness of anti-phishing systems in [6, 20, 21], and they showed that the design of a usable security system is a hard problem. The general consensus is that security indicators that rely on the user to make a correct decision tend to be ineffective in [2, 3, 7]. This is also supported by Sunshine et al., who concluded that warnings should be avoided when possible and decisions should be made for the user in an automated, under the hood fashion [17]. On the other hand, Whitten and Tygar showed that a careful design can lead to much more usable security in [20]. Brustoloni and Villamarin-Salomon presented promising results by trying to prevent users from getting habituated to existing warning dialogs and introducing morphic warning messages in [1].

3. OUR APPROACH

Our approach to addressing the SSLstripping attack is based on the use of visual security cues as a way to help overcome the semantic barrier between the user's mental model and the system's processing model. Specifically, the visual security cues will be displayed in login form fields so that users will be able to detect the attack based on them. Because of a preference of usability over security, visual security cues are often disregarded by users. Therefore, our challenge is to design and build a security solution that gives more value to users than the value of the effort required to use it. Unlike regular software systems, which users use because there is some kind of return for them, security is most of the times an add-on, and if the effort to include security is much higher than the perceived risk the user is exposed to, users will just ignore it. Hence it is of utmost importance to design a more easy-to-use and more intuitive, but less disruptive visual cue solution.

The design of our visual cues was motivated by the popular traffic light metaphor, which is simple to understand and familiar to most users. This is primary reason behind our adoption of the traffic light metaphor, and we strongly believe that it can be used to help simplify the decision making management for both lay and technically savvy users when they are about to submit their sensitive login credential. As we discussed previously, the attack is so effective that most users are not likely to be able to detect it. Therefore, we strongly believe that each security indicator simply represented by a single traffic light, red, yellow, or green and displayed inside of login form fields will better assist them to understand the current security situation that they are faced with and to make better, more informed decisions when they need to submit their sensitive information to a remote website. Motivated by this, our solution called security status

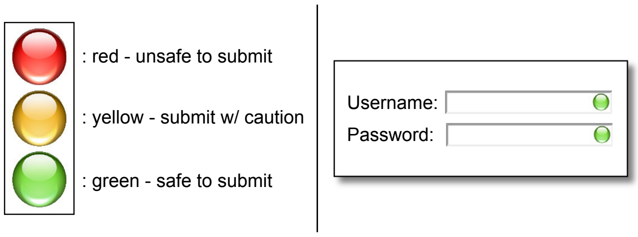


Figure 3: Our approach: security status light (SSLight) and how it is used in a login form

light (SSLight) adopts the three color design, as shown in Figure 3.

In following sections we first discuss an algorithm to display SSLight so that it can be used in judging whether login credentials are to be submitted securely or not. Next, we implement a browser extension that executes the algorithm and presents the result to the user.

3.1 SSLight

Before the user can be given some information regarding the website where she is about to log in, we need to be able to evaluate the security of this website. We developed an algorithm that will look at the web page source code and output an evaluation based on the comparison of the web page's address and the login form's action data, which has an URL where credentials will be submitted.

The first step is to identify if the page loaded by the web browser is already being accessed over SSL. If this is the case, we just need to verify that the action on the login form belongs to the same domain that is already secure, a situation that is true if the action is a relative path. This means the form will submit the login request to an address in the scope of the current secure connection. Hence, the SSLstripping is not possible and we return a positive evaluation (**Green light**). If the current page is not on a secure connection, we will assess the protocol being used in the form action. If we find that the form action URL is an insecure address, we immediately return a negative evaluation (**Red light**), which means it is unsafe to submit the login request.

In the scenario where the current page is not secure, but the form action is under the HTTPS protocol, we proceed with another round of analysis. Further analysis will first assess the certificate of the secure location referenced in the form action. If this is a self-signed or expired certificate, we will return a negative evaluation. Next, if the certificate proves not to be invalid, we compare the domain in the form action with the domain of the loaded page. If these two domains match, we return a positive evaluation, otherwise, we check whether the domain of the login form action is in a list of trusted login entities. If we cannot white-list the URL, we will issue an uncertain assessment (**Yellow light**) and delegate to the user, showing them the domain where they will be submitting their form. In an attack situation, a warning would appear in the webpage that previously did not raise any issues.

Our algorithm is outlined in pseudo code in the following listing:

```
EVALUATE-FORM()
if HTTPS-INITIAL()
    return green-signal
else
    if formAction ≠ https
        return red-signal
    else
        FURTHER-ANALYSIS()
```

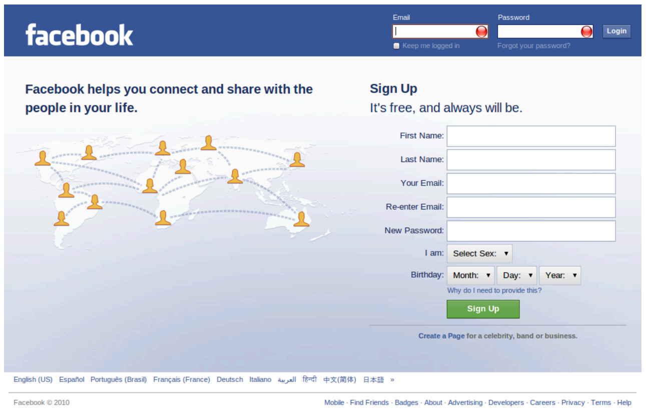
```
FURTHER-ANALYSIS()
if ¬VERIFY-SSL-CERTIFICATE()
    return red-signal
else
    if form.act.loc.hostname ≠ doc.loc.hostname
        if WHITE-LIST(form.act.loc.hostname)
            return green-signal
        else
            return yellow-signal
    else
        return green-signal
```

3.2 Browser Extension Implementation

SSLight was implemented as a browser extension on Google Chrome web browser using Javascript, HTML and CSS. This implementation should be able to be ported with the necessary adaptations to every other browser that allows extension development. The extension executes only after the page has been loaded, being the last layer of code to be executed on the rendered page. The algorithm we designed is run and after a conclusion is made, the result is presented to the user.

We chose to implement two different visual cue solutions. The first is SSLight present at all times in the login form fields in the loaded website. The SSLight will spin like the images in slot machines when the page is loaded and assessed, and will stop spinning and display an appropriate color when the assessment is made. The green light indicates that the form is secure, being submitted through a secure channel. The red light means the form is insecure and the yellow light represents a very small number of cases where we cannot make a definite assertion and delegate it to the user. This visual cue can be seen in Figure 4. Note that a tool tip warning message associated with each of the colored lights is not displayed here. However, they are used in our user study to deal with color-blinded people.

The other visual cue solution is to simply change the background color of the input boxes, either blinking or not, to a red color that alerts the user to danger. In the second solution, shown in Figure 5, nothing will be presented to the user if the assessment returns an output that signals a

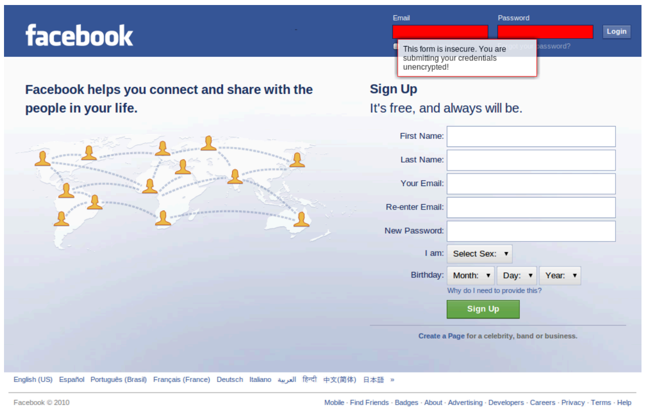

Facebook helps you connect and share with the people in your life.

Sign Up

It's free, and always will be.

facebook
Email
Password
Login
Keep me logged in
Forgot your password?
First Name:
Last Name:
Your Email:
Re-enter Email:
New Password:
I am:
Select Sex:
Birthday:
Month:
Day:
Year:
Why do I need to provide this?
Sign Up
Create a Page for a celebrity, band or business.
English (US) Español Português (Brasil) Français (France) Deutsch Italiano العربية हिन्दी 中文(简体) 日本語 »
Facebook © 2010
Mobile · Find Friends · Badges · About · Advertising · Developers · Careers · Privacy · Terms · Help

Figure 4: The SSLight browser extension


This form is insecure. You are submitting your credentials unencrypted!

Facebook helps you connect and share with the people in your life.

Sign Up

It's free, and always will be.

facebook
Email
Password
Login
Forgot your password?
First Name:
Last Name:
Your Email:
Re-enter Email:
New Password:
I am:
Select Sex:
Birthday:
Month:
Day:
Year:
Why do I need to provide this?
Sign Up
Create a Page for a celebrity, band or business.
English (US) Español Português (Brasil) Français (France) Deutsch Italiano العربية हिन्दी 中文(简体) 日本語 »
Facebook © 2010
Mobile · Find Friends · Badges · About · Advertising · Developers · Careers · Privacy · Terms · Help

Figure 5: The Blinking background browser extension

secure form submission. The extension configuration allows the user to choose which warning method should be used and if the non-disruptive tool tip style warning message is to be shown. By non-disruptive we mean that the warning will not require the user to stop her actions, i.e. writing the user name or password in the text fields, as opposed to warning methods that require the user to click a pop up window before any further action is performed.

4. USER STUDY

We conducted a user study[1] not only to test the effectiveness of our solution, but also to compare it against an existing solution, namely the classic pop-up window in Firefox. We also wanted to study how all of these solutions compare against the absence of any visual security cue. Finally, we wanted to test all of the warnings in the presence of the SSLstripping attack.

To test different scenarios that will be elaborated later, we devised four different user groups, each exposed to the attack and a different kind of warning or no warning at all. We wanted to investigate how effective our proposed solution is against an SSLstripping attack.

[1]This user study was approved by our university's IRB and conducted for one week at our university campus.

To avoid the framing effect, we did not want the users to be aware that we were testing their login behavior and their reaction to a security warning. In addition, we wanted the users to be exposed to the warning as an abnormality or exceptional condition. To achieve both the goals, we asked the users to perform different sets of tasks on their Facebook accounts. This made them use their real credentials and have tasks to perform that would make their behavior as close to reality as possible. We set up two machines, one being attacked, and another left untouched. The users were first asked to create a list of friends on their Facebook account and then add three of their friends to that list. This first set of tasks was performed on the untouched machine where the credential submission happened over HTTPS. Next, the users were asked to remove the very same list they had just created. This time around, when they tried to log in, a security warning would be displayed. It was what the users decided to do at this point that we were interested in.

Since we did not want to expose the users to any real danger, i.e. setting up a real SSLstripping attack where user credentials are sent unencrypted over the network and could be read by anyone sniffing it, we prevented an actual login from occurring on the second machine and redirected the users to the exit survey instead. The users who chose not to continue after noticing the security warning, were asked to login with no credential information, and therefore also redirected to our exit survey. Before the beginning of the study we defined a set of hypotheses that we wanted to have tested. The hypotheses we developed are:

1. **(General awareness of secure form submission)** Users will know the difference between encrypted and unencrypted credential submission when they log in to a website.

2. **(Effectiveness of SSLstripping, given the awareness)** Users will not notice any difference when they submit their credential to log in to a website under a successful SSLstripping attack.

3. **(Unhelpfulness of an existing method)** Users will submit their credential, even when warned with the classic pop-up alert message for unencrypted data submission.

4. **(Helpfulness of visual cue-based methods)** Users will not submit their credential, when warned with new visual cue-based approaches.

5. **(Effectiveness of different visual cue methods)** There is no difference in the submission ratio between blinking background and SSLight approaches.

The first hypothesis assumes that the average user is aware of the difference between secure and insecure form submission. In other words, the average user knows that a secure form submission encrypts the form data before sending it over the network. Given this knowledge, the second hypothesis tests whether users are able to detect the SSLstripping attack, meaning that users will not notice when a normally secure website submits their credentials unencrypted, the consequence of being under the SSLstripping attack. We hypothesize that the average user will not be able to detect the attack. The third hypothesis tests the assumption that the current warning method, a pop-up window with security warning message, will be ineffective to prevent the attack and users will still submit their credentials, despite the warning that the current page is submitting credentials unencrypted and despite the fact that in the previous login activity there was no warning. The last two hypotheses are related to our proposed approach. We hypothesize that our approach will be more effective than the traditional approach, and also that our two different visual cues will have identical effects.

4.1 Experimental Design

We defined four separate user groups, each of which is exposed to a different warning scenario:

- Group 1: Exposed to the attack with no warning
- Group 2: Exposed to the attack with the standard pop-up warning dialog
- Group 3: Exposed to the attack with the SSLight warning in the login form fields
- Group 4: Exposed to the attack with the blinking background in the login form fields

The users were asked to act as if they were using their own machines, in that all decisions they made should be the same as if they were being made on their own private computers. Security was never explicitly mentioned. Even though we initially thought the "make all decisions as if this was your machine" could bring too much focus into security issues, the results showed us that this was not the case. To avoid making the user aware of our real purpose, the experiment script was carefully designed so that the users thought they were being tested on the features of the website being accessed and not on the security of the login action itself. Users were assigned randomly to one of the user groups and given instructions. The instruction sets were identical for all the users except the username with which they were to login to the test machines. Each of the four groups was assigned a different username[2] corresponding to a different setup on the two test machines.

4.2 Sample

In the design of our experiment, we performed a power analysis to determine the minimum sample size that we would require to test our hypotheses. We chose an error of 0.05 and a power of 0.8, common among such experiments, and determined a minimum sample size of 19 subjects per study group, adding up to a total of 76 subjects across the four user groups.

In our study, volunteers were elicited by using fliers posted around our University campus and offering an iPod Touch as a prize to be drawn among the participants. A total of 106 individuals replied to our request, from which 5 did not perform the study and one submitted an empty exit survey form, which was discarded. A total of 100 valid submissions were obtained, 25 individuals per group, therefore above our minimum sample size and enough to test our hypotheses. The subjects who participated in our study mostly came from our university. The demographic details will be presented in the beginning of Section 5.

[2]Note that the users used their own username and password to login to facebook however.

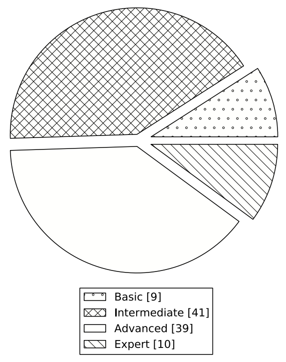


Figure 6: Distribution of computer usage expertise among study subjects.

4.3 Exit Survey

The last step asked to the participants of our study was to answer a brief exit survey that included questions on the overall understanding of a secure form submission and its consequences, the presence of warnings in browsers and the user opinion on the different warning mechanisms. Even though each participant was exposed to only one scenario in the study, they were shown all the three warnings in the exit survey and asked to classify them in a 5 point Likert scale. The users were also asked to select their favorite warning method, out of the methods presented and an "other" option, that would allow users to write any other method of insecure form submission warning they would prefer over the presented possibilities. The users were also asked to quantify their concern with the security of their credential submission to a website. We used a nine-point Likert scale for this end. Finally, the users were asked to answer a few questions on their information, such as gender, education and computer experience level.

5. USER STUDY RESULTS

Our sample includes 100 individuals, whose computer expertise is on Figure 6 and education level is on Figure 7. Our subjects have a high education rate, with all having completed at least high school and the gross of participants being undergraduate college students. Relative to computer usage experience, the individuals in our sample are very sophisticated, with only 9 claiming they have basic skills, 39 and 41 claiming they have intermediate or advanced skills and 10 rating themselves as expert computer users.

5.1 Hypothesis 1: General Awareness of Secure Form Submission

This hypothesis was tested by asking all the users about their understanding of secure form submission in the exit survey. On the exit survey, we asked users to select all the statements from a list that applied to unencrypted form submissions. The summary of results is on Table 1. The statements are ordered in decreasing order on the number of users that selected that statement as applicable to an insecure form submission.

As it can be seen from the results, 89% of the subjects are aware that an insecure form submission exposes their

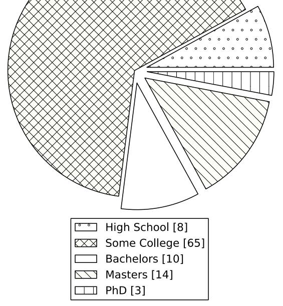


Figure 7: Study subjects education level.

Option	Positive Answer Count	95% CI
Your username and password can be seen by an attacker connected to your local network	89	0.811 - 0.944
Your browser does not encrypt your password	66	0.558 - 0.752
The encryption of your network is weak	42	0.322 - 0.523
The website to which you are submitting form will sell your information	32	0.23 - 0.421

Table 1: Unencrypted form submission danger awareness

credentials to eavesdroppers on their local network, while only 66% are aware that credentials submitted insecurely are not encrypted by their browsers. While 42% of the subjects think an insecure form submission is caused by the network they are connected to, 32% believe an insecure form would be the cause of their private information being leaked by a service to third party entities, which is not dependent upon the submission security, but the service being logged in to. From these results, our first hypothesis is confirmed, and we can assume that the average user is aware of the dangers of submitting data over the network using forms over plain http.

5.2 Hypothesis 2: Effectiveness of SSLstripping attack

On our second hypothesis, we assumed that the SSLstripping attack would be highly effective. This means that the average user, being exposed to the attack, will not be able to detect it and will submit the form as if it was secure. To test this hypothesis, we had the first group of users exposed to the attack and not shown any warning. Facebook has the login form submitted to an HTTPS address by default, and after the login is performed, the user gets redirected to an HTTP address again. We postulate that the average user will not be aware of this situation, but there is still the possibility that a savvy user would be able to detect the attack by inspecting the source code.

Of the twenty-five users in the first user group, none of them detected the attack and all submitted the login form. This 100% effectiveness in the sample translates into an interval from 88.7% to 100% in the population, with 95% confidence. This interval confirms our hypothesis and allows us to assume that without any added security mechanisms, the SSLstripping attack will be highly effective against the average user.

	Submit	Not submit	Fisher's Exact P
No Warning	25	0	1
Pop-up Window	24	1	

Table 2: Comparison of having the Pop-up Window and having no warning

5.3 Hypothesis 3: Unhelpfulness of Existing Method

The method for warning users of an insecure form submission and the dangers associated with it has traditionally been a pop-up window that appears after they entered their credentials in the form fields and clicked the submit button associated with the form. This method was present in Netscape Navigator and is currently an optional feature on Mozilla Firefox and Internet Explorer.

We hypothesized that the average user, when presented with this warning, would ignore it and proceed with the login operation. In our experiment the users were requested to login to Facebook.com on two different machines. On the first of these machines no warning was displayed, but on the second machine, the one under attack, Firefox would warn the users of the insecure form submission via the optional pop-up window. Table 2 shows the results we obtained, compared with the results obtained for the absence of any warning and the statistical analysis of difference between these two groups.

With only one user out of 25 refusing to continue when presented with the pop-up window, the statistical test reveals that there is no meaningful difference between having no warning and having the pop-up window.

5.4 Hypothesis 4: Helpfulness of Visual Cue Methods

The main goal of our user study was to investigate the effectiveness of the novel visual cue methods we proposed to notify the user of a threatening situation, namely the submission of a form containing sensitive credential information over an unencrypted communication channel. The previous two hypotheses were established as assumptions that we made during the design and development of our approach. While testing our solution, we also needed to establish if our assumptions were correct, which was verified by failing to reject the null hypotheses in Section 5.1 and Section 5.2. Having support for these two assumptions, we can now compare them to the results obtained with the two user groups exposed to our proposed warning methods. The summary of our results and of the statistical significance of the difference between the different user groups is shown in Table 3 and 4.

	Submit	Not submit	Fisher's Exact P
Pop-up Window	24	1	0.011
SSLight	16	9	

Table 3: Comparison of having the Pop-up Window and our SSLight Solution

	Submit	Not submit	Fisher's Exact P
Pop-up Window	24	1	0.000
Blinking	8	17	

Table 4: Comparison of having a Pop-up Window and our Blinking Background Solution

We can see from Table 3 that 16 out of the 25 users in the SSLight user group submitted the form, while 9 refused to continue. This shows a significant difference to the proportion found in the pop-up window user group and confirms hypothesis 4 for our first approach. The second solution we proposed had only 8 out of the 25 users in the user group submitting the form. A total of 17 users refused to continue in the presence of this warning method. The statistical difference between this group, and the group exposed to the classic pop-up window is even more evident than that of our first solution. The results are summarized in Table 4.

Hypothesis 4 is therefore confirmed for both our solutions, meaning that they are indeed more effective than the traditional approach to warning a user of an insecure form submission, as we failed to reject both the null hypotheses regarding the visual cues.

5.5 Hypothesis 5: Effectiveness of Different Visual Cue Methods

Our fifth and last hypothesis stated that both of our visual cue solutions would be equally effective, with no statistical difference between the ratio of users that ignored it and the ratio of users that refused to submit their credentials. The results, summarized in Table 5, show that there is indeed a statistical difference and therefore the null hypothesis is rejected. This means that the blinking background was statistically more effective than the SSLight solution. Failing to support hypothesis 5 indicates that our visual cue mechanisms are not equally effective, even though both are more effective than the classic approach.

6. OTHER RESULTS AND DISCUSSION

6.1 Preferred Methods

	Submit	Not submit	Fisher's Exact P
SSLight	16	9	0.046
Blinking	8	17	

Table 5: Comparison of having our SSLight Solution or our Blinking Background Solution

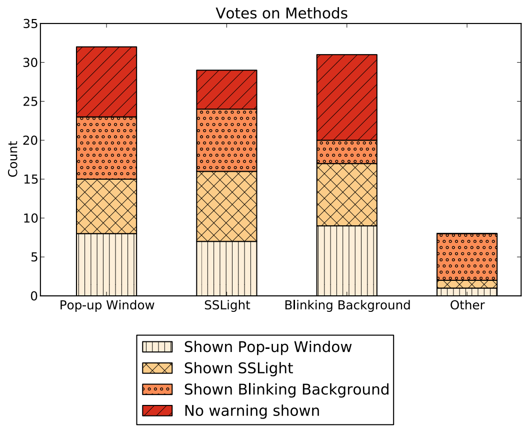


Figure 8: The number of votes attributed to each of the different methods, stacked according to the user group that cast the vote.

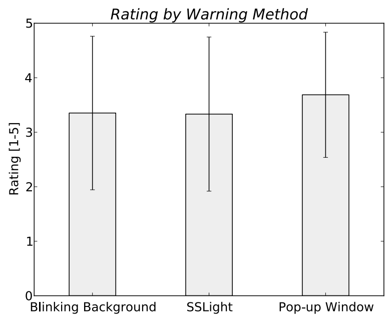


Figure 9: The average rating attributed to each of the methods.

One of the questions included in our exit survey was relative to all the methods presented to the different groups. The question asked each individual in our user study to choose a favorite warning method out of the three methods used: pop-up window, SSLight and blinking background. A final option tagged "other" accompanied with an empty text field was also provided to allow users the freedom of not choosing one of the mentioned solutions, but still answer the question.

Figure 8 shows the compilation of the answers to this question. The chart shows not only the total number of votes each method received, but also how the different user groups voted across the different methods. It is interesting to notice that the only group that had no votes in the "other" option was the group that was exposed to no warning.

In addition to asking the users to select their preferred method, we asked them to rate each of the methods presented in the survey. The summary of the results is in Figure 9. The graph shows the average rating of each method, along with the standard deviation. These results show that there is not a significant difference between the user rating of each method.

6.2 Reaction to Warnings in the Past

	Yes	No
Noticed Warning in the Past	84	16
Deactivated Warning	36	47

Table 6: Distribution of users who have noticed insecure form submission warnings in the past and, among those who have, the fraction of those who opted out of them and those who did not.

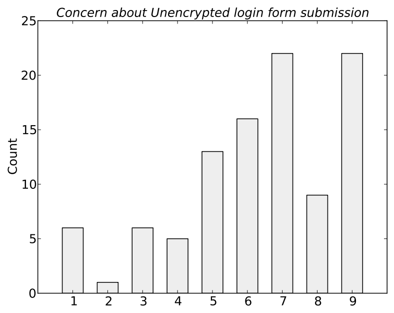


Figure 10: Distribution of the concern about unencrypted login form submission.

Trying to understand the past user behavior in face of the classic pop-up window, we asked users if they had ever seen any warning about submitting form insecurely over the network. The results, summarized in Table 6, show that 84% of the users have in fact noticed an insecure form submission warning in the past. Of these, 43% say they opted out of the warnings and 56% claim they did not. The 1% missing is explained by a user who said he had seen a warning in the past but did not answer whether he opted out of it or not.

It is, however, important to note that at the time that this study was conducted, none of the major web browsers had a default insecure form submission warning.

6.3 User Concern

One of the first questions we asked in the exit survey was relative to the subject's personal concern about insecure form submissions. On a scale of 1 to 9, being 1 not concerned and 9 very concerned, the average rating was 6.31, with a standard deviation of 2.246. It is then reasonable to assume the average user is somewhat concerned with the risks of submitting the credentials in plain text over the network. The distribution of the different ratings can be seen on Figure 10, where we can see an higher incidence on votes on the higher half than the lower half.

6.4 Alternate Methods

When we asked the users to select their favorite warning method, we also left room for the possibility that the users would rather not have any of the proposed methods, or even no warning method at all. We included an "other" option which also allowed them to write what would that other method be. The results collected from the users who chose "other" and provided further explanation include "a color-coded warning in the url bar", "padlock-flashing red", and "status message below the address bar".

6.5 Potential Attacks

Considering that the SSLstripping attack is a MITM attack, where the attacker can change the content of the webpage being rendered by the browser, it is reasonable to assume that the attacker can change more than just the action attribute of a form to remove the 's' from an address starting with "https". We took special consideration to the scenario where the attacker assumes our extension is present and may try to circumvent it.

Our first concern is that an attacker could create an overlaying html component that could sit on top of the original form element, or just our warning, thus giving the user a false sense of trust by always displaying a positive warning. To avoid this attack scenario, our warning is rendered not on the form element itself, but on a separate layer placed on top of the form element. An alternate solution would involve having the shape of the warning, a circle in the case of SSLight, to be defined by the user. This means that the attacker would have to correctly guess the shape chosen by the user for the warning in addition to the remainder of the attack. For the blinking background approach, we could allow the user to choose its own color scheme, forcing the attacker to have the burden of discovering this information. We chose to implement an overlaying component as opposed to custom warnings, as it requires less effort from the user.

Another possible tweak to the attack that could be used to circumvent our approach is to include in the page a script that would change the action URL of the form only after our algorithm has finished running. This would mean that although the extension may have seen the original action address and it was deemed to be secure, the injected code could change it and the user would actually submit the form to an insecure location. To address this issue, our extension makes use of the existing Google Chrome extension API to verify the request after it has been made and double check the submission address. If at this point a risk is identified, the user will be warned and prompted for a confirmation.

It is noteworthy that both the scenarios above would require changes to the original SSLstripping attack that need to be tailored to individual web pages. The attack would no longer be as general and therefore not as powerful as the original SSLstripping attack.

7. CONCLUSION AND FUTURE WORK

In this paper we presented a novel approach to thwarting the SSLstripping attack by using visual security cues. Our empirical study clearly shows that the proposed solutions are more effective and efficient in preventing the SSLstripping attack than the classic pop-up window method. However, our approach is by no means complete. For our immediate future work, we will investigate how to improve the effectiveness of the SSLight solution. Specifically, the proposed SSLight is based only on color and text message. We will investigate how to add additional factors such as symbol or position to enhance its effectiveness. Another future work is related to our experimental design. Our study used a sample consisting mainly of higher education students, a demographic that does not represent the average user accurately. Gathering a more representative sample poses a bigger challenge that we are trying to address. There is also the fact that the results obtained by our approaches could stem directly from its novelty alone. To circumvent this, we are working on another round of data collection that will require longer and more frequent interactions to exclude the novelty as a factor for the good results, thereby studying the effect of user habituation. Lastly, the usual rules that apply to a desktop browser are not applicable to a browser running on a smartphone, for example. We see the emergence of long solved security flaws in new devices, as is the case of the recent browser user interface spoofing in iPhone. These new exploitation opportunities come from the lack of consideration for security aspects in software development, specially in user interface design of mobile applications. We will investigate how our approach can be applied to this in the near future.

ACKNOWLEDGEMENT

This work was partially supported at the Secure Computing Laboratory at New Mexico Tech by the grant from the National Science Foundation (NSF-IIS-0916875).

8. REFERENCES

[1] J. C. Brustoloni and R. Villamarín-Salomón. Improving security decisions with polymorphic and audited dialogs. In *Proceedings of the 3rd symposium on usable privacy and security*, SOUPS '07, pages 76–85, New York, NY, USA, 2007. ACM.

[2] L. F. Cranor. A framework for reasoning about the human in the loop. In *Proceedings of the 1st Conference on Usability, Psychology, and Security*, pages 1:1–1:15, Berkeley, CA, USA, 2008. USENIX Association.

[3] R. Dhamija, J. D. Tygar, and M. Hearst. Why phishing works. In *Proceedings of the SIGCHI conference on Human Factors in computing systems*, CHI '06, pages 581–590, New York, NY, USA, 2006. ACM.

[4] T. Dierks and C. Allen. The TLS protocol version 1.0. RFC, 1999. http://www.ietf.org/rfc/rfc2246.txt.

[5] T. Dierks and E. Rescorla. The transport layer security (TLS) protocol version 1.2. RFC, 2008. http://tools.ietf.org/html/rfc5246.

[6] S. Egelman, L. F. Cranor, and J. Hong. You've been warned: an empirical study of the effectiveness of web browser phishing warnings. In *Proceeding of the twenty-sixth annual SIGCHI conference on Human factors in computing systems*, CHI '08, pages 1065–1074, New York, NY, USA, 2008. ACM.

[7] B. J. Fogg, J. Marshall, O. Laraki, A. Osipovich, C. Varma, N. Fang, J. Paul, A. Rangnekar, J. Shon, P. Swani, and M. Treinen. What makes web sites credible?: a report on a large quantitative study. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, CHI '01, pages 61–68, New York, NY, USA, 2001. ACM.

[8] P. Gutmann and I. Grigg. Security usability. *Security Privacy, IEEE*, 3(4):56 – 58, 2005.

[9] C. Jackson and A. Barth. Forcehttps: protecting high-security web sites from network attacks. In *Proceeding of the 17th international conference on World Wide Web*, WWW '08, pages 525–534, New York, NY, USA, 2008. ACM.

[10] C. Jackson, D. Simon, D. Tan, and A. Barth. An evaluation of extended validation and picture-in-picture phishing attacks. In S. Dietrich and R. Dhamija, editors, *Financial Cryptography and Data Security*, volume 4886 of *Lecture Notes in Computer Science*, pages 281–293. Springer Berlin / Heidelberg, 2007.

[11] M. Marlinspike. New tricks for defeating SSL in practice. In *Red Hat 2009*, Washington DC, 2009.

[12] N. Nikiforakis, Y. Younan, and W. Joosen. HProxy: Client-side detection of SSL stripping attacks. In C. Kreibich and M. Jahnke, editors, *Detection of Intrusions and Malware, and Vulnerability Assessment*, volume 6201 of *Lecture Notes in Computer Science*, pages 200–218. Springer Berlin / Heidelberg, 2010.

[13] S. Schechter, R. Dhamija, A. Ozment, and I. Fischer. The emperor's new security indicators. In *Security and Privacy, 2007. SP '07. IEEE Symposium on*, pages 51 –65, May 2007.

[14] B. Schneier. Semantic attacks: The third wave of network attacks. *CryptoGram Newsletter*, October 2000.

[15] J. Sobey, R. Biddle, P. C. Oorschot, and A. S. Patrick. Exploring user reactions to new browser cues for extended validation certificates. In *Proceedings of the 13th European Symposium on Research in Computer Security: Computer Security*, ESORICS '08, pages 411–427, Berlin, Heidelberg, 2008. Springer-Verlag.

[16] D. Stebila. Reinforcing bad behaviour: the misuse of security indicators on popular websites. In *Proc. 22nd Australasian Conf. on Computer-Human Interaction (OZCHI) 2010*. ACM, 2010. In press.

[17] J. Sunshine, S. Egelman, H. Almuhimedi, N. Atri, and L. F. Cranor. Crying wolf: an empirical study of SSL warning effectiveness. In *Proceedings of the 18th conference on USENIX security symposium*, SSYM'09, pages 399–416, Berkeley, CA, USA, 2009. USENIX Association.

[18] T. Whalen and K. M. Inkpen. Gathering evidence: use of visual security cues in web browsers. In *Proceedings of Graphics Interface 2005*, GI '05, pages 137–144, School of Computer Science, University of Waterloo, Waterloo, Ontario, Canada, 2005. Canadian Human-Computer Communications Society.

[19] A. Whitten. *Making Security Usable*. PhD thesis, Carnegie Mellon University, 5000 Forbes Avenue Pittsburgh, PA 15213-3890, May 2004.

[20] A. Whitten and J. D. Tygar. Why johnny can't encrypt. In *In Proceedings of the 8th USENIX Security Symposium*, 1999.

[21] M. Wu, R. C. Miller, and S. L. Garfinkel. Do security toolbars actually prevent phishing attacks? In *Proceedings of the SIGCHI conference on Human Factors in computing systems*, CHI '06, pages 601–610, New York, NY, USA, 2006. ACM.

[22] M. Zurko. User-centered security: stepping up to the grand challenge. In *Computer Security Applications Conference, 21st Annual*, pages 14 pp. –202, 2005.

AdSentry: Comprehensive and Flexible Confinement of JavaScript-based Advertisements

Xinshu Dong[†], Minh Tran[‡], Zhenkai Liang[†], Xuxian Jiang[‡]

[†] Department of Computer Science
National University of Singapore
{xdong, liangzk}@comp.nus.edu.sg

[‡] Department of Computer Science
North Carolina State University
{mqtran, xuxian_jiang}@ncsu.edu

ABSTRACT

Internet advertising is one of the most popular online business models. JavaScript-based advertisements (ads) are often directly embedded in a web publisher's page to display ads relevant to users (e.g., by checking the user's browser environment and page content). However, as third-party code, the ads pose a significant threat to user privacy. Worse, malicious ads can exploit browser vulnerabilities to compromise users' machines and install malware. To protect users from these threats, we propose AdSentry, a comprehensive confinement solution for JavaScript-based advertisements. The crux of our approach is to use a shadow JavaScript engine to sandbox untrusted ads. In addition, AdSentry enables flexible regulation on ad script behaviors by completely mediating its access to the web page (including its DOM) without limiting the JavaScript functionality exposed to the ads. Our solution allows both web publishers and end users to specify access control policies to confine ads' behaviors. We have implemented a proof-of-concept prototype of AdSentry that transparently supports the Mozilla Firefox browser. Our experiments with a number of ads-related attacks successfully demonstrate its practicality and effectiveness. The performance measurement indicates that our system incurs a small performance overhead.

1. INTRODUCTION

Internet advertising is one of the most popular business models of today's Internet companies. For example, more than 96% of Google's revenue is from Internet advertising [15]. In Internet advertising, web site owners or web publishers include advertisements (or "ads") from advertisers in their pages, and get paid by advertisers when users view and click on these ads.

To increase the likelihood for users to click on the ads, advertisers commonly use (JavaScript) code in ads to check a user's browser environment to select advertisements that are believed to be more attractive to the users. As third-party code, these ads unfortunately pose great security threats to both web applications and the underlying operating systems. For example, such ads require close integration with the displayed page contents, which may leak users' private data [28] and break web applications' integrity. Worse, malicious ads can further exploit software vulnerabilities in web browsers to launch drive-by downloads and surreptitiously install malware on users' machines. A recent research shows that "about 1.3 million malicious ads are being viewed online everyday, most pushing drive-by downloads and fake security software" [33].

To mitigate the threats from untrusted ads, a number of solutions have been recently proposed. They address the threats to user privacy and web application integrity by sandboxing JavaScript ads through functionality restriction or isolation [2, 4, 10, 13, 17, 19, 23, 26, 27, 30, 35, 40, 45, 49]. However, they cannot block ads from triggering drive-by downloads, which have been "persistently" plaguing online users as one of the main attack mechanisms. In addition, these solutions are not flexible in controlling behaviors of JavaScript advertisements: the allowed access of an ad must be decided before it starts to run. In the face of these limitations, there is a need for an integrated solution that can not only flexibly regulate the ad access to various web contents, but also effectively block drive-by downloads from malicious ads.

In this paper, we present the design, implementation and evaluation of AdSentry, a comprehensive and flexible isolation framework to confine JavaScript-based advertisements. Instead of supporting only a subset of JavaScript functionality or isolating the ad execution through significant changes to web pages, AdSentry provides a *shadow JavaScript engine* for untrusted ad execution. The purpose of having a shadow JavaScript engine is to ensure that the ad will not affect the host web content without proper control, thus protecting user privacy and the integrity of web applications. More importantly, it provides the control in a transparent manner and still exposes the full spectrum of JavaScript functionality to the untrusted ads. Meanwhile, to block possible drive-by downloads and preserve the integrity of the host system, the shadow JavaScript engine is strictly sandboxed.

By design, AdSentry effectively mediates all accesses made by untrusted ads to the web application. We stress that the shadow JavaScript engine (for the ad execution) by default cannot access the original page DOM (Document Object Model). To accommodate legitimate accesses by ads to some part of the page content, AdSentry transparently interposes related DOM accesses from the ads. For every such access, AdSentry checks its legitimacy (according to a given access control policy) and, if benign, redirects it to the original page DOM to substantiate the access. Our framework is flexible in allowing both web publishers and end users to specify or customize the access control policies for ads, as well as allowing dynamically changing policy after an ad starts to execute.

We have implemented a proof-of-concept AdSentry prototype. The shadow JavaScript engine implementation is based on the open-source Mozilla SpiderMonkey. Its execution is strictly sandboxed with Native Client [48], which has demonstrated its effectiveness

ACSAC '11 Dec. 5-9, 2011, Orlando, Florida USA
Copyright 2011 ACM 978-1-4503-0672-0/11/12 ...$10.00.

in confining third-party code with high efficiency and reliability. Our development experience further indicates that AdSentry is a generic framework that can be conveniently implemented as a regular browser extension without requiring the modification of the browser code. Our evaluation results with a number of ad-related exploits show that AdSentry is effective in successfully blocking all of them. The performance evaluation shows that the protection is achieved with a low overhead.

Contributions.

We identified the critical need for a confinement solution to prevent ads from threatening the privacy and integrity of user data as well as the integrity of users' computing systems. To summarize, this paper makes the following contributions:

- AdSentry provides a comprehensive isolation framework to confine untrusted ads on web pages. Its comprehensiveness is achieved by not only regulating the ad access to the original web application (or web contents), but also effectively confining the ad execution in a strongly sandboxed environment to block possible drive-by downloads.

- Our solution preserves the original execution environment for third-party scripts. Unless otherwise specified by access control policies, AdSentry does not alter the execution order of different scripts on the page even after certain ad scripts are sandboxed by our solution. Sandboxed ads scripts also have full access to global JavaScript objects created or overwritten by other scripts outside the sandbox, if allowed by access control policies.

- Our solution allows for flexible mediation of each DOM access from untrusted ads. It is also flexible in allowing both web publishers and end users to specify access control policies for ads. We highlight that it is important to empower users with full control as they can choose how to protect the viewed web pages from ads (according to their own requirements and running environments) and remain confident in protecting the integrity of their host systems.

- We have implemented a prototype that transparently supports modern Mozilla Firefox browsers. Our prototyping and evaluation results with real-world examples demonstrate its practicality and effectiveness.

The rest of this paper is organized as follows: Section 2 provides an overview on Internet advertising. Sections 3 and 4 present our system design and implementation. Section 5 presents detailed evaluation results. Section 6 discusses limitations of our approach and suggests future improvement. Finally, Section 7 describes related work, and Section 8 concludes the paper.

2. PROBLEM OVERVIEW

In Internet advertising, advertisers pay web publishers directly to display their ads on these web sites, or more often, pay advertising networks to get their ads displayed on popular sites, easily reaching out to a large amount of audiences. Moreover, advertisers usually allow web sites or advertising networks to dynamically decide what kind of ads to display to their visitors, based on the web contents users are viewing. This behavior is called "targeting" of ads. It makes Internet advertising more relevant and presumably more helpful to visitors. The profit of web publishers hosting ads can be calculated by different revenue models, including measuring how many times the ads are displayed, how many visitors have seen the ads, or how many times the ads have been clicked by visitors, etc. [47]

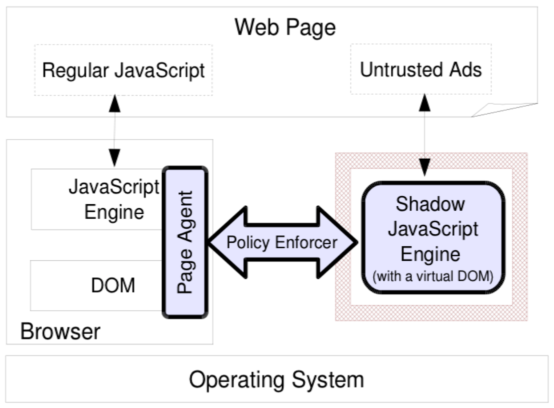


Figure 1: An architecture overview of AdSentry. The core components of AdSentry are highlighted in the figure.

Internet advertising brings in new challenges to web security and privacy. One possible way for publishers to include advertisements is to completely isolate them in separate iframes. However, such a complete isolation makes advertisement targeting impossible. As a result, third-party ads are often included in `<script>` elements, so they have the same privilege as other JavaScript on the web page.

In this paper, we focus on such third-party JavaScript-based advertisements (ads) that are deployed on web pages. These ads are hosted outside web publishers' servers, but included as JavaScript on the web pages. If some of them become malicious, they may abuse their privileges in accessing web application data for various purposes, such as leaking confidential user information and issuing unauthorized transactions. Moreover, they may exploit software vulnerabilities in browsers to take over the users' systems.

Our goal in this work is to comprehensively confine these untrusted JavaScript ads and effectively protect users' privacy and the integrity of both web applications and the users' computer systems.

3. SYSTEM DESIGN

To effectively confine untrusted ads, we have four design goals, i.e., *comprehensiveness*, *flexibility*, *transparency*, and *efficiency*. By comprehensiveness, we aim to provide an integrated scheme that not only regulates ad access to the host web page, but also contains malicious ads from launching drive-by downloads. The flexibility requirement allows both web publishers and end users to specify access control policy for ads. Users can also dynamically change access control decisions based on application runtime states. The transparency goal requires no modification to the browser for the support and preserves the timing of the JavaScript behaviors in the web applications. The transparency requirement ensures that the current billing model of ads is not affected. Actually, it is a stronger requirement than simply requiring no changes to ads billing. Also, the proposed solution needs to be efficient in introducing low performance or maintaining a similar level of user experience.

Following these design goals, we have developed a novel ad isolation framework called AdSentry, whose overall architecture is shown in Figure 1. In essence, AdSentry provides a shadow JavaScript engine to confine untrusted ads. This shadow JavaScript engine by default has no direct access to the original browser environment and the operating system. Therefore, the ads can be fully confined, and the host web page and OS will remain intact even if the ads are malicious. To meet the transparency requirement, the shadow JavaScript engine can be seamlessly integrated into current browsers through the standard browser's extension application programming interfaces (APIs), i.e., no browser modification will be necessary.

With the introduction of a shadow JavaScript engine, AdSentry essentially works with two JavaScript engines: the untrusted ads run inside the shadow engine while the rest (normal) JavaScript in the web page runs as usual in the default engine. We point out that an ad may have legitimate reasons to access certain web content (e.g., for the purpose of advertisement targeting). To accommodate these requests, AdSentry provides a virtualized *DOM* to the shadow JavaScript engine. The virtual DOM has all the standard DOM interfaces, including XMLHttpRequest, so page accesses made by ads running in the shadow engine will be received by the virtual DOM. When the virtual DOM is being accessed, it will relay the access to the *page agent* in the browser through a *policy enforcer*. The policy enforcer will decide whether a page access is allowed by users' security policies. If yes, the page agent proceeds with the access request on behalf of the isolated ad, and returns the results back to the isolated ad through the virtual DOM. If not, the access will be blocked to protect the integrity of the web page.

Besides virtualizing the DOM access for untrusted ads, AdSentry also sandboxes the ad execution within the shadow engine, preventing them from compromising users' operating systems.

It is important to note that AdSentry is transparent to web pages by automatically dispatching ads to the shadow engine and seamlessly supporting their accesses. As a result, the billing model of ads is not affected. AdSentry preserves the original execution timings of all JavaScript in the web page, including ads scripts running in the shadow JavaScript engine. This is a key advantage of AdSentry over iframe-base isolation techniques, such as AdJail. This ensures that the behaviors of the applications and user experience will not be altered unless for security concerns, making AdSentry applicable to securing a wider class of untrusted JavaScript code in web applications.

3.1 Shadow JavaScript Engine

By introducing a shadow JavaScript engine to host untrusted ads, AdSentry allows us to achieve the comprehensiveness goal: resilience against exploits to browsers themselves and protection for the confidentiality and integrity of web application data. As mentioned earlier, the shadow JavaScript engine is executed inside a Native Client (NaCl) [48] sandbox. There are two reasons why we choose to build our system on top of NaCl. First, NaCl sandbox has been shown to be secure against code injection attacks with minor performance overhead. Second, NaCl has been supported on a number of platforms, such as x86, x86-64 and ARM [41], which can be very helpful for adopting AdSentry by end users, especially with the rising popularity of the Google Chrome browser and the Chrome OS that ships NaCl as one built-in component.

Like the normal JavaScript engine in the current browser, the shadow JavaScript engine is shared across web pages in the browser. To distinguish different ads from different pages, each ad will be assigned a unique identification number. With that in place, when an ad needs to be executed, AdSentry sends its JavaScript and the ad's identification number to the shadow engine. To preserve the original execution timing of the web page, the browser waits until the ad's JavaScript finishes in the shadow engine, in the same way that the original browser JavaScript engine handles its execution.

Specifically, once the sandboxed JavaScript engine receives a JavaScript to execute, it creates a new JavaScript context for the ad with the associated identification number. A virtualized DOM will be initiated to contain a set of global objects, which are then made accessible to the JavaScript context. The virtual DOM has all standard DOM interfaces, but each interface is simply a stub that forwards the access to it to the page agent in the browser. After the initialization, the shadow JavaScript engine starts executing the received JavaScript. If the script accesses a particular DOM interface, the access is intercepted by the virtual DOM, which in turn communicates with the page agent to handle the access request.

3.2 Page Agent

The intercepted DOM access requests from the virtual DOM are forwarded to the page agent, which resides on the same page with the web application. After the verification from the policy enforcer, the page agent will perform the requested DOM access on behalf of the ad. If the request is to create or modify DOM element(s), the page agent takes special care to capture all resulting JavaScript executions and forwards them back to the shadow engine for processing. For instance, when a user clicks on a button created by an ad, the triggered `onclick()` function call will be captured and executed in the shadow engine.

The communication between the virtual DOM and the page agent is in the form of message passing. When the page agent receives a message requesting a DOM access from the shadow JavaScript engine, it extracts the access from the message, and processes it in the context of the original web page. The page agent then sends the result back to the virtual DOM, and in turn, to the shadow JavaScript engine, completing the access made by the confined ad.

AdSentry also naturally regulates access to HTTP requests from untrusted ads. Specifically, ads may initiate HTTP requests by either generating new DOM elements (that have already been controlled by the page agent), or by directly initiating XMLHttpRequest. As XMLHttpRequest is not part of the JavaScript engine, but is provided by the virtual DOM, the invocation to XMLHttpRequest is also regulated by our system.

In order to ensure that the relayed DOM access is transparent to the executing ad, we need to address a few issues – some of them come from innate JavaScript features.

Dynamically generated JavaScript.

Ads can insert a new piece of JavaScript into a web page. The new JavaScript must also be executed in the same shadow JavaScript engine. Otherwise, the newly generated JavaScript can escape the isolation of AdSentry.

There are several ways for ads to introduce new JavaScript into the original web page. Examples include abusing `document.write` or setting the `innerHTML` attribute of an element. Accordingly, whenever AdSentry receives a message from the shadow JavaScript engine requesting to invoke such DOM interfaces, the request is interpreted and all newly introduced JavaScript is properly flagged to ensure they will execute in the shadow JavaScript engine. We detail this solution in Section 4.

Timers and event listeners.

One interesting challenge comes from the support of asynchronous events, such as timers, where ads register callback routines to be executed later. Such callback routines need to be executed in the shadow JavaScript engine. To handle these asynchronous events, AdSentry dynamically creates a stub as the corresponding event handler in the web page. This stub will notify or invoke the true callback routine in the shadow JavaScript engine.

Unfortunately, as asynchronous events occur unexpectedly, the notifying message sent to the shadow JavaScript engine may arrive in the middle of the execution of some other DOM accesses, which causes an undesirable race condition. To avoid that, the messages from asynchronous events will be separately marked and temporally buffered by the shadow JavaScript engine. These messages will then be processed after the ongoing DOM accesses are finished.

Anonymous functions.

The JavaScript language supports anonymous functions. For example, the following code snippet creates an anonymous function with a function body *alert(0).*

```
window.addEventListener("click",
         function () { alert(0); },
         false);
```

As an ad may create these anonymous functions, we need to isolate them properly. Particularly, if these anonymous functions are being used as event listeners, they should be invoked within the shadow JavaScript engine when the corresponding events occur. Unfortunately, anonymous functions are represented as native function objects in the JavaScript engine, rather than strings of JavaScript code. Therefore, we cannot handle them in the same way as we do for JavaScript code on the page.

To address this problem, in our system, when the shadow JavaScript engine executes a statement that creates an anonymous function, we record the function's internal identification number, associate that number with the related DOM access, and forward it to the page agent. When the page agent receives the message at the real DOM side, it dynamically composes a new JavaScript function whose task is just to send a message containing the identification number of the anonymous function to the shadow JavaScript engine. After that, it assigns the newly composed function as the argument to the event listener. When the event occurs, the newly composed function will be invoked (at the real DOM side) to send a message to the shadow JavaScript engine and ask it to run the anonymous function with the specified identification number.

3.3 Policy Enforcer

By confining the shadow JavaScript engine within a sandboxed environment, our system effectively blocks possible drive-by downloads that target the underlying JavaScript engines (more concrete examples will be shown in Section 5). In the meantime, it is important to point out that the sandbox itself does not provide any guarantee on the confidentiality or integrity of the web application. As a result, it needs to work in concert with the policy enforcer to achieve this goal. Specifically, the policy enforcer checks the requests intercepted by the virtual DOM according to a given user security policy. Only if allowed by the enforced security policy, the request will then be forwarded to the page agent for processing.

As mentioned earlier, our system allows both web publishers and end users to customize the access policy for ads. Specifically, for web publishers, as they can simply change the web page content, they may choose to wrap the ad and confine its execution in the shadow JavaScript engine. For end users, our current system leverages Adblock Plus [34] to automatically identify ads and confine them with a customized JavaScript wrapper. We will present the details as well as the supported policies in the next section.

4. IMPLEMENTATION

We have implemented a proof-of-concept prototype of AdSentry based on the browser extension support of Firefox, and it is implemented and tested in Mozilla Firefox 3.5.8. Our implementation of the shadow JavaScript engine is based on Mozilla SpiderMonkey version 1.8.0. The virtual DOM support is generated with a code generator of 770 SLOC in perl. On the browser side, the other two components (i.e., the policy enforcer and the page agent), are implemented entirely in the JavaScript language. These two components add about 3100 SLOC.

4.1 Specifying Advertisement Scripts

AdSentry is flexible in deployment. It allows both web publishers and end users to specify the scripts to be executed in the sandbox. The intuitive way is to associate an attribute with the script indicating it is an advertisement script. However, this solution requires the browser to be modified to recognize the attribute. In AdSentry, we provide a function `sandboxAds`. It takes the body of an ad or the URL of an ad script, and notifies AdSentry to execute it in isolation.

Therefore, to use AdSentry, web publishers can process the advertisement with the function `sandboxAds`, as illustrated by the following example, where the last argument indicates whether the first argument is the URL (true) or the body of an ad script (false).

```
<script>
  sandboxAds('http://ads.com/ad.js',
             id, true)
</script>
```

AdSentry also provides the option to end users by automatically identifying ads instances at the client side. It uses Adblock Plus [34] to identify ads and automatically processes them with `sandboxAds`.

4.2 Shadow JavaScript Runtime and Virtual DOM

We use NaCl to sandbox the SpiderMonkey JavaScript engine. We found this process relatively straightforward. However, the main challenge comes from the extension we make to the original JavaScript engine. Specifically, to enable ads running in the JavaScript environment to access related page content (e.g., for ad rendering), there is a need to provide virtual DOM objects. In our prototype, a virtual DOM is made available to the JavaScript engine in the form of a tree of objects. The root of this tree is called the global object. [1] In the case of a web page, the global object is the `window` object. This global object has a number of properties, including global JavaScript variables and functions, such as the `document` object, the `location` object, and the `eval` function. With this tree structure, all other virtual DOM objects are also properties of their parent objects.

We obtain a standard DOM structure from the standard DOM specifications [44], construct virtual DOM objects and expose them to the shadow JavaScript engine as host objects. More specifically, the virtual DOM for SpiderMonkey is generated in the following steps: 1) Create a new `JSRuntime` object and set up initial configurations and in runtime, create `JSContexts` for the execution of ads scripts. 2) Create a `JSClass` for each class of DOM objects. 3) Specify properties and member functions for each `JSClass`. 4) Implement property and function accessor methods for each `JSClass`, most of which will invoke one of the centralized access handling functions, respectively. 5) Implement the centralized access handling functions for virtual DOM accesses. These functions will then relay the access to the page agent (on the browser side). To relay the access, they also perform other tasks, such as preparing arguments to actual DOM function calls, looking

[1]Note that the concept of the global object here is different from that of global objects in a JavaScript program.

for anonymous functions, buffering event listener code for later execution, etc. These functions interpose each and every access from ads to the real DOM. 6) Create instances of standard objects from `JSClass` definitions, starting from the `window` global object. For non-global objects, we will specify their parent objects during the creation to form the tree structure.

Considering the large number of virtual DOM objects we need to construct and the associated tree structure we need to maintain, we have a code generator in place to automate the above steps for all virtual DOM objects. The code generator reads in an XML file that specifies the DOM tree objects and structures, and then generates an output file that embeds the JavaScript engine and sets up its host environment with the virtual DOM.

4.3 Page Agent

To facilitate the communication between the shadow JavaScript engine and the page agent, we define a simple message format for data exchange. The format is summarized as follows:

```
msg::= command data
command::=  script | callFunc | getProp
          | setProp | return
data::= <text>
```

Each message contains a *command* field and a related *data* field. Our prototype has defined five different commands: a `script` command is used to notify the page agent that an ad script needs to be sent to the shadow engine for execution. Upon receiving the message, the shadow engine will prepare the runtime environment and then start executing it. During execution, it will intercept any DOM access from the ad script and based on the type of access, translate it into three other types of messages to the page agent: `callFunc` for function invocations, `getProp` for property retrievals, and `setProp` for property (re)initialization. Finally, a `return` command carries the results in the message body, i.e., the *data* field.

The page agent extends the Firefox browser through its standard extension interfaces. We create a Firefox extension, which monitors the dispatched message events notified by `sandboxAds`. Specifically, following the above message format, if a *script* command is received, it parses the message stored in the event object, and communicates with the sandbox. We stress that web pages cannot directly communicate with the sandbox, and all communications are done via the Firefox extension. During the ad execution, if it needs to access a DOM object, the sandbox intercepts it and encapsulates the access by sending a message to the page agent requesting a DOM access. Here, the DOM access is meant for the access of the real web page and the extension cannot evaluate it in its own execution environment.

There are two possible approaches for our extension to evaluate the intended DOM access in the web page context. The first approach is straightforward: simply posting a message (or dispatching a custom event) to the web page. After receiving it, the web page can then evaluate the requested DOM access (encoded in the message or event). However, message passing is asynchronous, which allows other JavaScript on the same web page to preempt the execution of the current ad script. This kind of preemption may cause serious problems as it alters the original execution order of different scripts on the page. For example, scripts may have dependency on each other, and a premature execution of a later script may fail if the dependent script has not been executed. As another example, `document.write` is normally executed before a web page is loaded. If it's executed after a page is loaded, it creates a new page, completely eliminating the original one. To execute sandboxed scripts normally, AdSentry should not alter the original execution order of scripts on the web page, so this first approach is not suitable here.

The second approach is to implement the communication between the web page and the extension like a function call. Mozilla Firefox provides a mechanism for extensions to evaluate JavaScript code in web pages' privileges, called `evalInSandbox` [32]. In our prototype, we leverage this method to call a function in the context of the web page that contains the ad script, which in turn evaluates the DOM access being requested, and returns the result to the extension. After that, the extension sends it back to the sandbox via a pipe. By doing so, when an ad script is being executed, we can ensure the JavaScript engine in the original browser environment is always in one of the three states: a) waiting for messages from the sandbox; b) executing our script in the extension; or c) executing the message processing function in the host web page while our extension is waiting for the return. As a result, no other scripts on the web page could preempt the current execution of ads script.

Consequently, our implementation is based on the second communication approach. More details are discussed below.

Concurrent ads scripts.

AdSentry supports processing multiple ads scripts concurrently. To avoid mix-ups of ad scripts from different web pages, our browser extension maintains a message queue to ensure that only one ad script is being processed at any point of time. Each message sent to the shadow engine is marked with an identification number, enabling the engine to evaluate each ad script in its own JavaScript context. When evaluating DOM accesses requested by the sandbox, AdSentry also makes sure the accesses will be evaluated in the same page that originally contains the ad script being executed.

Object maps.

The communication mechanisms implemented in AdSentry are text-based, but in some cases we need to pass objects as parameter or return values. This is achieved by maintaining object maps at both the page agent and the shadow JavaScript engine, and only communicating the objects' indices in the messages. Before a JavaScript object is to be communicated to the other end, it is checked against the local object map. If it already exists in the map, its index is returned; otherwise, it is inserted into the map with its new index returned. Then in the message sent, the index of the object is included, rather than the object's real data. Next time when a message is received from the other end containing an object index, the object is restored by querying its index from the local object map.

Parameter buffering.

AdSentry enforces security policies on the result of JavaScript actions, which will be described in Subsection 4.4. One possible way to bypass our access control policy enforcement is to insert content into the web page piece by piece. For example, instead of calling

```
document.write("<scr" + "ipt> some script <"
               + "/scr" + "ipt>");
```

malicious ad script may attempt to avoid being detected by inserting a `script` element like the following

```
document.write("<scr");
document.write("ipt> some script <");
document.write("/scr");
document.write("ipt>");
```

This way, checks on parameters to each individual DOM function call would not detect that a new `script` element is being inserted.

To prevent such misuses, AdSentry buffers such consecutive function calls by not sending them one by one to the shadow JavaScript engine for execution, but finally replace them with a single call with the entire piece of content being inserted, which is checked by the access control policy enforcer as normal.

4.4 Access Control Policy Enforcement

To regulate the communication between the host web page and the confined ad script, our policy enforcer acts as a moderator. Any communication between the two parties needs to be approved according to a given policy. AdSentry is flexible in allowing both web publishers and end users to specify the access control policies for ads.

AdSentry has a default policy. The default policy disallows any JavaScript code originated from ads to run in the host web page. In other words, all untrusted scripts will be guaranteed to be only executed inside the shadow JavaScript engine. To enforce that, we examine all incoming messages from the sandbox, distinguish page updates containing dynamic JavaScript content versus static HTML, and then handle them accordingly.

Specifically, for the static HTML content, our system first normalizes the HTML into the corresponding XML format and then serializes the XML back to HTML before processing. The HTML code is widely known as badly formed, to the point that badly written code is often called “tag soup” [6]. Also, all major browsers have permissive parsing behaviors by supporting a rendering mode called “quirks mode” beside the “standards mode” [5]. These browser quirks have many negative implications, one of which is that malicious attacker can embed JavaScript code inside a malformed fragment of HTML code. To strive a balance between security and the support of potential browser quirks, we took three phases for parsing HTML code. First, we attempt to reformat the code by correcting popular mistakes in web authoring. For instance, we close all open tags and correct all improperly nested tags. Second, we leverage the XML parser in the web browser to parse this reformatted code into a XML model. Note that a malformed HTML is considered dangerous and will be rejected by our parser. Since XML parser is strictly standard-compliant, any surviving formation will bear no ambiguity. Finally, we serialize this XML model back to HTML code before handing to the page agent for further processing.

For the JavaScript dynamically generated by ads scripts, we install wrappers that request the sandbox to run the dynamic JavaScript code. In other words, all untrusted scripts are guaranteed to execute inside the shadow page, not the real page. In our prototype, we apply the code wrapping based on the above XML model. Specifically, we leverage the XML XPath facility available in most browsers to traverse the XML model tree and inspect the enclosed nodes. We first query for patterns of dynamic code on the model. These patterns of dynamic code include event handlers such as `onclick()`, as well as related JavaScript functions such as `addEventListener()`, `setTimeout()` and `setInterval()`. The resulting node set will then be properly wrapped or transformed. In our prototype, we have installed wrappers on all 32 possible vectors of dynamic code and ensure that no potentially malicious code will ever be injected to the real page. As an example, the following code snippet

```
setTimeout(' slideAd(10,100); ', slideDelay);
```

will be transformed into the following code fragment:

```
setTimeout(' sandboxAds(" slideAd(10,
       100);", id, false); ', slideDelay);
```

To further ensure the privacy of sensitive user data in the web page, we allow users to configure the data to be shared with the script. As mentioned earlier, we do not copy all the content of the real DOM to the virtual DOM. Instead, we choose to interpose on every access to the virtual DOM from the untrusted ad and subject it for policy verification. As such, users can decide to be extremely cautious with certain kind of ads, and block any read access from the ad to the entire page. On the other extreme, a user might want to trust certain ads, and allow free accesses to the real DOM content. In addition to the above two policies, a user is also allowed to specify a policy that blocks accesses to the `document.cookie` object or mandates that ad can only read from its own elements and not the surrounding content. Moreover, an ad can be prohibited from appearing outside of the allocated region of the web page (by stating the allowed values of `width, height` and `overflow` property of ad elements). This is helpful to thwart some types of phishing attacks. In fact, as a comprehensive isolation framework, our system provides a mediation capability that can accommodate existing access control polices [23] for ads. And both web publishers and end users can take the advantage of the same capability to enforce security policy on ad behaviors.

AdSentry also enables end users to dynamically specify access control policy with tools during the execution of web applications. In addition, AdSentry leverages a customized version of Adblock Plus [34] to automatically identify and wrap ads scripts on web pages.

5. EVALUATION

In this section, we evaluate the functionality and performance of AdSentry. In particular, we have conducted four sets of experiments. The first one is based on real-world browser exploits to evaluate AdSentry’s defense against drive-by download attacks. The second one is to test its resilience against malicious attempts that inject JavaScript into web applications. The third one is to evaluate AdSentry’s protection of privacy against rogue information-stealing ads; The fourth one is to measure the performance overhead. Our experiments were conducted on a Dell E8400 workstation with a Core 2 Duo CPU (3GHz 6 MB L2 Cache) and 4GB of RAM. The system runs Ubuntu 9.10 and we its the default web browser – Mozilla Firefox 3.5.8 – for our experiments.

5.1 Browser Exploits

To evaluate the effectiveness of AdSentry in sandboxing ads, we conducted experiments with a few real-world exploitations, obtained from existing research work [25] as well as vulnerability databases [1, 31]. All the exploits we tested with caused the vulnerable versions of the Firefox browser to crash during our experiments. They are all marked as critical by Mozilla developers, and can be further crafted to launch severe attacks such as drive-by download.

Our experiments are summarized in Table 1. The eight examples exploit the vulnerabilities in the SpiderMonkey JavaScript engine. Most of them are various instances of buffer overflow or memory corruption attacks, and they could lead to arbitrary code execution. With AdSentry installed in the vulnerable versions of the Firefox browser (in our experiments, we used Firefox 3.0 and Firefox 3.5 for corresponding exploits), each of the exploits was successfully contained by the shadow JavaScript engine. This confirmed and demonstrated one of our design goals that we would like to run untrusted ads scripts in an isolated environment so that even in the worst case, they would not crash the entire web browser. As AdSentry sandboxes the JavaScript engine, so any memory attack against vulnerabilities in the JavaScript engine would be contained by the sandbox.

Bugzilla ID	Attack Behavior	Outcome
426520	Browser crashed by memory corruption with crafted XML namespace	Contained by shadow JS engine
454704	Browser crashed by exploiting a vulnerability of XPCSafeJSObjectWrapper	Contained by shadow JS engine
465980	Browser crashed by pushing to an array of length exceeding limit	Contained by shadow JS engine
493281	Browser crashed by stack corruption starting at unknown symbol	Contained by shadow JS engine
503286	Browser crashed by exploiting a vulnerability of Escape()'s return value	Contained by shadow JS engine
507292	Browser crashed by incorrect upvar access on trace involving top-level scripts	Contained by shadow JS engine
561031	Browser crashed by overwriting jump offset	Contained by shadow JS engine
615657	Browser crashed by buffer overflow due to incorrect copying of upvarMap.vector	Contained by shadow JS engine

Table 1: AdSentry evaluation using browser exploits

Scenario	Attack Vector	Attack Behavior	Outcome	Description
1	Direct code injection	Inject script	Blocked	Denied by the default policy
2	Browser parsing quirk	Malformed $< img >$ tag	Blocked	Rejected by message normalization
3	Browser parsing quirk	Malformed $< script >$ tag	Blocked	Rejected by message normalization
4	Browser parsing quirk	Malformed $< script >$ tag	Blocked	Rejected by message normalization
5	Browser parsing quirk	Malformed $< b >$ tag	Blocked	Rejected by message normalization
6	Browser parsing quirk	Malformed $< script >$ tag	Blocked	Rejected by message normalization
7	Browser parsing quirk	Malformed $< iframe >$ tag	Blocked	Rejected by message normalization

Table 2: AdSentry evaluation using JavaScript injection attacks

5.2 Script Injection by Ads

In our second experiment, we evaluated the effectiveness of our default policy in preventing untrusted code from being injected from the ad to the web page. In particular, we examined the XSS Cheat Sheet [7] and identified a number of cases that can successfully result in injecting JavaScript from the ad into the web page for execution. We confirmed the successful injection and execution in the default Firefox without AdSentry being installed. During our experiments, we explicitly cleared the browser's cache between each step.

Our results are shown in Table 2. The first one is a direct attempt to include an external JavaScript to execute in the web page while the other six exploit numerous parsing quirks [7]. Such attacks are created to execute a simple script that displays a message box "hacked!" The use of browser parsing quirks reflects the current trend [24] in part because they are much harder to repair without breaking compatibilities with legacy web applications. This was blocked by the default policy in AdSentry that direct injection of scripts into the web page is disallowed.

For the rest examples, we use the second scenario as the representative. Specifically, in the second scenario, the attempt is to exploit a parsing quirk by embedding a `<script>` tag as literal text inside a `<img>` tag, which will cause the browser to interpret the text string as JavaScript code, thus causing an injection:

```
<IMG """><SCRIPT>alert("XSS")</SCRIPT>">
```

The related code snippet is shown above. It contains three pairs of double-quotes, encapsulating different parts of the text. If a parser were properly implemented, there would be three literal strings: an empty string `""`, the second string `"><SCRIPT>alert("` and the last string `")</SCRIPT>"`. These three strings are orphaned as they are not assigned to any property of the tag and therefore should be discarded. As such, the entire tag should simply collapse to `<IMG XSS>`, which can also be disregarded. However, this is not the case in most modern browsers. In fact, existing browsers tend to be very permissive in their parsing behavior [5]. For instance, we observed that Firefox interpreted `<IMG """>` as the first tag and `<SCRIPT>alert("XSS")</SCRIPT>` as the second tag; the remaining `">` was accepted as plain text and displayed as is. As a result, the "malicious" code `alert("XSS")` was executed. This attempt was blocked because of the normalization through the standard-compliant XML in our system. We successfully detected this malformed HTML content and substituted it with the benign static text "Script Injection Blocked."

5.3 Privacy Protection

In our third set of experiments, we test our system from the privacy perspective. In particular, it has been known that third-party JavaScript can violate user privacy in various ways. Examples include cookie stealing, location hijacking, history sniffing, and behavior tracking [20]. In our experiments, we evaluated AdSentry with a synthesized ad that simulates the above information-stealing behaviors.

In particular, the synthesized ad is developed to perform all these four types of behaviors: The cookie stealing is implemented to access the `cookie` property of `document` object; The location is hijacked by setting the `location` property of `window` (or `document`); The previously browsed URLs are sniffed by obtaining the color of the populated hyperlinks, which can be done by invoking the `getPropertyValue` function of the `ComputedCSSStyleDeclaration` object (with the argument "color") [2]; Behavior tracking is achieved by registering related event listeners of interested elements, such as `onclick`, `onmouseover`, etc.

AdSentry successfully detected each of the above four types of behaviors. For the first two types, our system simply denies the read access to the `document.cookie` and the write access to the `window.location` and `document.location`. For the third type of ad behavior, it is detected by monitoring any invocations to the related `getPropertyValue` function. For behavior tracking, AdSentry refused the registration of callback routines of those elements if the ad does not own them.

We stress that our privacy protection enforcement does not suffer from JavaScript object and property aliasing problems. This is because the access is intercepted by the virtual DOM that, when invoked, has already resolved all object and property aliasing, if any.

Moreover, we also evaluated the user experience of AdSentry using 15 popular website with ads, shown in Table 3. The embedded ads are automatically recognized by the Adblock Plus extension

[2]Recent browsers return the same computed styles for visited and unvisited links.

Web site	Properties of ads
www.msn.com	Ads on different domain of same company
www.aol.com	Ads on content distribution network (CDN)
www.livejournal.com	Ad network DoubleClick
espn.go.com	Ad network DoubleClick
www.cnet.com	Ads on different domain of same company
imageshark.us	Ad network Google
www.nytimes.com	Ad network Checkm8
www.ehow.com	Ad network YieldManager
sourceforge.net	Ad network DoubleClick
www.reference.com	Ad network DoubleClick
www.dailymail.co.uk	Ad network DoubleClick
www.guardian.co.uk	Ad network Google
www.gmx.net	Ad network Uimserv
yfrog.com	Ad network Rubicon Project
www.comcast.net	Ad network Yahoo!

Table 3: Web sites used in user experience evaluation

and then transparently confined with AdSentry. To allow users to interactively specify security policies, we integrate a Firefox extension called Firebug [3] and extend it with a pop-up menu that can be triggered with a right mouse click. Specifically, we use the Firebug to visually capture available screen regions and for a selected region, a right mouse click will activate the pop-up menu. From the menu, a user will be shown the list of ads (grouped by domains) currently embedded in the current page and can then choose which ad can have a read access to the chosen screen region or can register call-back routines (e.g., event listeners). By default, these ads are only allowed to read their own elements, not the surrounding areas. Users can also specify new policies during run time, which will overwrite existing ones if necessary.

Our experiments did not find any suspicious information-stealing behavior for these websites.

5.4 Performance Evaluation

In order to assess the performance overhead, we conducted experiments to measure the page load overhead. We picked up four typical ads, one from each of the top four ad networks. We created a test page for each ad and ran the test page with and without AdSentry. Each experiment was repeated for 20 times, and the average results were recorded.

Our results are shown in Table 4. Overall, AdSentry incurs small overhead. The relative overhead ranges from 3.03% in MSN Ad Network ad to 4.96% in Google Adsense ad. We observed that a typical ad might only infrequently access DOM namespace, which might attribute to the low overhead. From another perspective, the relative overhead can be low because ad content such as images are often dynamically loaded from a remote server, this process experiences network round trip delay that is typically much more significant than local computation time in web browsers. Also, to improve responsiveness, modern browsers typically start rendering any elements immediately once they are available. Therefore a user may not notice the difference in the speed of ad loading time at all. In other words, this pipelining of the rendering process contributes to masking the delay that may be experienced by any single element in a web page.

In addition to the above real ads, we also measure the time needed to initialize our sandbox. Our results show that it takes 31 ms to initialize and set up the sandbox. Though it is lightweight, we expect opportunities still remain to reduce the time by further optimizing the JavaScript engine and NaCl sandbox. Finally, we evaluate a round-trip communication delay for a virtual DOM access. Without our system, it typically took 0.001 ms for the ad to finish the reading of a particular DOM property. When being confined, it will take 0.59 ms. This is expected as it needs to cross the sandbox boundary and go through the normalization for policy verification. Note that this overhead will be effectively amortized in real-world scenarios – as demonstrated in the four real ads.

6. DISCUSSION

In this section, we discuss the limitation of AdSentry and future work. First, our current work focuses on the JavaScript-based advertisements and has not yet explored the support of other types of advertisements. In particular, Flash technology is another popular way to write and display ads, which still remains to be investigated how flash-based ads can be supported.

Second, AdSentry protects the browsers from attacks exploiting vulnerabilities of the JavaScript engine, but it is not designed to prevent attacks to other browser components, such as the HTML rendering engine. If the malicious HTML segment is dynamically generated by JavaScript code, AdSentry's policy engine can mitigate the attack by the HTML normalization and signature-based attack blocking. A more general solution is to extend our solution to isolate other components of the browser.

Finally, we will continue to work on improving AdSentry's compatibility with JavaScript on a web page. Our prototype implementation is able to handle typical JavaScript advertisements, which has limited ways in accessing other parts of the web page. However, third-party JavaScript code in general has much tighter integration with the rest of the web page. As our future work, we will improve the support for transparently isolating a wider class of JavaScript code in web applications. It will also be interesting to investigate possible ways (e.g., in software testing) that automatically test AdSentry's compatibility with a broader set of web applications.

7. RELATED WORK

In this section, we discuss existing work that mitigates threats from untrusted web content embedded into web applications, including those compromising user data, web application integrity as well as users' operating systems.

Drive-by download prevention.

Drive-by downloads are serious threats to web and host security [37,38]. BLADE [25] proposes a detection system for drive-by download exploits. This type of attacks has recently received lots of attention. For example, heap-spraying attacks can pre-populate a large heap space with attack code and a software bug can be exploited to redirect execution flow to the heap sprays (with attack code). In addition, several systems [11, 12, 39] have been proposed to leverage specific memory characteristics of these attacks to identify them and prevent browsers from being exploited. WebShield [21] proposes a middlebox framework that processes page contents in a shadow browser, and transforms DOM updates to the client browser to reflect DOM changes there. As a result, drive-by downloads can be detected at the middlebox without affecting the client browser. Other existing sandbox and isolation solutions [14, 22] can also be used to protect the operating system against drive-by download attacks. Compared to AdSentry, solutions in this category are not designed to protect user privacy and web application integrity from malicious JavaScript ads.

Isolation in web browsers.

Several recent research projects [9,16,46] attempt to achieve better browser security architecture by running different browser components in isolated environments. The Google Chrome browser

Performance Test	**with AdSentry** (ms)	**without AdSentry** (ms)	**Overhead** $(\%)$
Google Adsense Rendering	381	363	4.96
DoubleClick Ad Rendering	601	578	3.98
MSN Ad Rendering	1224	1188	3.03
Yahoo Ad Rendering	1539	1475	4.34

Table 4: Runtime page load overhead of AdSentry

also uses a sandbox to isolate browser components and protect the operating system [8, 29]. The IBOS [42] system steps further by designing a secure architecture for both the operating system and the web browser altogether, minimizing default sharing and trust between software components. However, they do not support isolating JavaScript ads from the rest of web applications, while AdSentry executes untrusted ads scripts in a separate and sandboxed environment from trusted scripts, mediating every access from ads to web applications.

Web application integrity protection.

To prevent tightly-integrated third-party JavaScript from affecting the integrity of web application, one type of solutions [2,10,13, 17,26,27] restricts the "dangerous" functionality of JavaScript. For example, ADsafe [10] only allows ads to use a safe subset of the JavaScript functionality. It removes dangerous JavaScript features, such as global variables, `eval`, `this`, and `with`. ADsafety [36] proposes a lightweight and efficient verification for JavaScript sandboxes, and has been successfully applied to ADsafe. Another line of solutions [4,19,35,40,49] protects web application against JavaScript ads through code transformation, enforcing policies against malicious JavaScript at runtime. Similarly, ConScript [30] introduces aspect into JavaScript language to enforce users' security rules. MashupOS [45] proposes new script integration primitives reflecting different trust relationships between the integrator and the mashup content provider. Besides enabling web publishers to protect their web applications, AdSentry also allows end users to flexibly specify access control policies according to their own requirements.

AdJail [23] addresses the privacy and web application integrity threat from ads by isolating them into an iframe-based sandbox. Using a separate origin in the sandbox, AdJail leverages browser's native origin-based protection to isolate ads. It is a solution for publishers to isolated third-party ads. Compared to AdSentry, AdJail assumes the ads on a web page are relatively independent and do not have tight dependencies with the page environment. For example, ad scripts cannot access global JavaScript objects defined or overwritten by other trusted scripts in the same hosting page. AdSentry transparently supports tight dependency between ads and the host page, without significant modification of the web page. It also provides flexible control of behaviors of JavaScript ads.

In addition, solutions in this category cannot prevent malicious ads from exploiting browser vulnerabilities.

Privacy protection.

One of users' major concern about JavaScript ads is privacy. Privad [18] proposes a solution to protect users' privacy by making users anonymous to the advertisers and publishers, but it does not prevent users' data from being used by the ad script, which may implicitly leak our user data. Adnostic [43] uses a browser extension to perform ad targeting, selecting ads to display from a larger set of ads sent by the advertisement network. Compared to AdSentry, both solutions only focus on protecting users' privacy, and do not address the ad's threat to integrity of web applications and the underlying operating system.

8. CONCLUSION

JavaScript-based advertisements are ubiquitous on the Internet. They pose threats to the privacy and integrity of web applications, as well as security of operating systems. In this paper, we present the design, implementation, and evaluation of AdSentry, a comprehensive and flexible framework to confine untrusted JavaScript advertisements. AdSentry not only separates the untrusted ad execution in a shadow JavaScript engine, but also mediates their access to the main page with access control policies, which can be specified by both web publishers and end users. We have implemented a Linux-based prototype of AdSentry that supports current Firefox browsers. Our experiments with a number of ad-related exploits show that AdSentry is effective in blocking these attacks. Our performance evaluation shows that the comprehensive protection is achieved with a small performance overhead.

Acknowledgments

We thank Michael Wright for his help in investigation and implementation. We also thank the anonymous reviewers for their insightful comments that helped improve the presentation of this paper. This work is supported in part by an NUS Young Investigator Award R-252-000-378-101, the US Army Research Office (ARO) under grant W911NF-08-1-0105 managed by NCSU Secure Open Systems Initiative (SOSI), and the US National Science Foundation (NSF) under Grants 0855297, 0855036, 0910767, and 0952640. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the ARO and the NSF.

9. REFERENCES

[1] Common vulnerabilities and exposures. `http://cve.mitre.org/`.
[2] FBJS (Facebook JavaScript). `http://developers.facebook.com/docs/fbjs/`.
[3] Firebug. Web Development Evolved. `http://getfirebug.com/`.
[4] Google Caja. `http://code.google.com/p/google-caja/`.
[5] Quirk Mode. `http://en.wikipedia.org/wiki/Quirk_mode`.
[6] Tag Soup. `http://en.wikipedia.org/wiki/Tag_soup`.
[7] XSS (Cross Site Scripting) Cheat Sheet. `http://ha.ckers.org/xss.html`.
[8] A. Barth, C. Jackson, C. Reis, and The Google Chrome Team. The security architecture of the chromium browser. `http://seclab.stanford.edu/websec/chromium/`.
[9] R. S. Cox, S. D. Gribble, H. M. Levy, and J. G. Hansen. A safety-oriented platform for web applications. In *IEEE Symposium on Security and Privacy*, 2006.
[10] D. Crockford. ADsafe. `http://www.adsafe.org/`.
[11] Y. Ding, T. Wei, T. Wang, Z. Liang, and W. Zou. Heap Taichi: Exploiting Memory Allocation Granularity In Heap-Spraying Attacks. In *Proceedings of the 26th Annual Computer Security Applications Conference (ACSAC)*, 2010.
[12] M. Egele, P. Wurzinger, C. Kruegel, and E. Kirda. Defending browser against drive-by downloads: Mitigating heap-srpaying code injection

attacks. In *Proceedings of the 6th Conference on Detection of Intrusions and Malware & Vulnerability Assessment (DIMVA)*, 2009.

[13] M. Finifter, J. Weinberger, and A. Barth. Preventing capability leaks in secure javascript subsets. In *Proc. of Network and Distributed System Security Symposium, 2010*.

[14] Goldberg, Wagner, Thomas, and Brewer. A Secure Environment for Untrusted Helper Applications: Confining the Wily Hacker. In *Proceedings of the 5th USENIX Security Symposium*, 1996.

[15] Google Inc. Google Fiscal Year 2010 Results, 2010. `http://investor.google.com/earnings/2010/Q4_google_earnings.html`.

[16] C. Grier, S. Tang, and S. King. Secure web browsing with the op web browser. In *IEEE Symposium on Security and Privacy*, 2008.

[17] S. Guarnieri and B. Livshits. Gatekeeper: mostly static enforcement of security and reliability policies for javascript code. In *Proceedings of the 18th conference on USENIX security symposium*, SSYM'09, pages 151–168, Berkeley, CA, USA, 2009. USENIX Association.

[18] S. Guha, B. Cheng, A. Reznichenko, H. Haddadi, and P. Francis. Privad: Rearchitecting Online Advertising for Privacy. Technical Report MPI-SWS-2009-004, Max Planck Institute for Software Systems, Germany, 2009.

[19] S. Isaacs and D. Manolescu. WebSandbox - Microsoft Live Labs. `http://websandbox.livelabs.com/`, 2009.

[20] D. Jang, R. Jhala, S. Lerner, and H. Shacham. An Empirical Study of Privacy-Violating Information Flows in JavaScript Web Applications. In *Proceedings of the 17th ACM Conference on Computer and Communications Security (CCS)*, 2010.

[21] Z. Li, T. Yi, Y. Cao, V. Rastogi, Y. Chen, B. Liu, and C. Sbisa. WebShield: Enabling various web defense techniques without client side modifications. In *Proceedings of the 18th Annual Network and Distributed System Security Symposium (NDSS)*, San Diego, CA, USA, 2011.

[22] Z. Liang, V. Venkatakrishnan, and R. Sekar. Isolated Program Execution: An Application Transparent Approach for Executing Untrusted Programs. In *Proceedings of the 19th Annual Computer Security Applications Conference (ACSAC)*, 2003.

[23] M. T. Louw, K. T. Ganesh, and V. Venkatakrishnan. AdJail: Practical Enforcement of Confidentiality and Integrity Policies on Web Advertisements. In *Proceedings of the 19th USENIX Security Symposium*, 2010.

[24] M. T. Louw and V. Venkatakrishnan. Blueprint: Robust Prevention of Cross-site Scripting Attacks for Existing Browsers. In *Proceedings of the 30th IEEE Symposium on Security and Privacy*, 2009.

[25] L. Lu, V. Yegneswaran, P. Porras, and W. Lee. BLADE: An Attack-Agnostic Approach for Preventing Drive-By Malware Infections. In *Proceedings of the 17th ACM Conference on Computer and Communications Security*, 2010.

[26] S. Maffeis, J. Mitchell, and A. Taly. Run-time enforcement of secure javascript subsets. In *Proc of W2SP'09*. IEEE, 2009.

[27] S. Maffeis and A. Taly. Language-based isolation of untrusted javascript. In *Proceedings of the 2009 22nd IEEE Computer Security Foundations Symposium*, pages 77–91, Washington, DC, USA, 2009. IEEE Computer Society.

[28] Matthew. Facebook's response to uproar over ads. `http://endofweb.co.uk/2009/07/facebook_ads_2/`.

[29] S. McCloud. The Chrome Comic Book, 2008. `http://www.google.com/googlebooks/chrome/index.html`.

[30] L. A. Meyerovich and B. Livshits. ConScript: Specifying and Enforcing Fine-Grained Security Policies for JavaScript in the Browser. In *Proceedings of the 31st IEEE Symposium on Security and Privacy*, 2010.

[31] Mozilla. Bugzilla@mozilla. `https://bugzilla.mozilla.org/`.

[32] Mozilla. Components.utils.evalInSandbox. `https://developer.mozilla.org/en/Components.utils.evalInSandbox`.

[33] R. Naraine. Research: 1.3 Million Malicious Ads Viewed Daily. `http://threatpost.com/en_us/blogs/research-13-million-malicious-ads-viewed-daily-051910`.

[34] W. Palant. Adblock Plus. `https://addons.mozilla.org/en-US/firefox/addon/adblock-plus/`.

[35] P. H. Phung, D. Sands, and A. Chudnov. Lightweight Self-protecting JavaScript. In *Proceedings of the 4th International Symposium on Information, Computer, and Communications Security*, ASIACCS '09, 2009.

[36] J. G. Politz, S. A. Eliopoulos, A. Guha, and S. Krishnamurthi. ADsafety type-based verification of javascript sandboxing. In *Proceedings of the 20th USENIX Security Symposium*, San Francisco, CA, USA, 2011.

[37] N. Provos, P. Mavrommatis, M. A. Rajab, and F. Monrose. All your iframes point to us. In *Proceedings of the 17th USENIX Security Symposium*, pages 1–15, 2008.

[38] N. Provos, D. McNamee, P. Mavrommatis, K. Wang, and N. Modadugu. The ghost in the browser analysis of web-based malware. In *Proceedings of the first conference on First Workshop on Hot Topics in Understanding Botnets*, pages 4–4, Berkeley, CA, USA, 2007. USENIX Association.

[39] P. Ratanaworabhan, B. Livshits, and B. Zorn. NOZZLE: A Defense Against Heap-spraying Code Injection Attacks. In *Proceedings of the 18th USENIX Security Symposium*, 2009.

[40] C. Reis, J. Dunagan, H. J. Wang, O. Dubrovsky, and S. Esmeir. BrowserShield: Vulnerability-Driven Filtering of Dynamic HTML. In *Proceedings of the 7th Symposium on Operating Systems Design and Implementation (OSDI)*, 2006.

[41] D. Sehr, R. Muth, C. Biffle, V. Khimenko, E. Pasko, K. Schimpf, B. Yee, and B. Chen. Adapting Software Fault Isolation to Contemporary CPU Architectures. In *Proceedings of the 19th USENIX Security Symposium*, 2010.

[42] S. Tang, H. Mai, and S. T. King. Trust and protection in the illinois browser operating system. In *Proceedings of the 9th USENIX conference on Operating systems design and implementation*, OSDI'10, pages 1–8, Berkeley, CA, USA, 2010. USENIX Association.

[43] V. Toubiana, A. Narayanan, and D. Boneh. Adnostic: Privacy Preserving Targeted Advertising. In *Proceedings of the 17th Annual Network and Distributed System Security Symposium (NDSS)*, 2010.

[44] W3C. Document Object Model (DOM) Specifications. `http://www.w3.org/DOM/DOMTR`.

[45] H. J. Wang, X. Fan, J. Howell, and C. Jackson. Protection and communication abstractions for web browsers in mashupos. In *Proceedings of twenty-first ACM SIGOPS symposium on Operating systems principles*, SOSP '07, pages 1–16, New York, NY, USA, 2007. ACM.

[46] H. J. Wang, C. Grier, A. Moshchuk, S. T. King, P. Choudhury, and H. Venter. The multi-principal os construction of the gazelle web browser. In *Proceedings of the 18th conference on USENIX security symposium*, SSYM'09, pages 417–432, Berkeley, CA, USA, 2009. USENIX Association.

[47] Wikipedia. Online advertising - Revenue models. `http://en.wikipedia.org/wiki/Online_advertising#Revenue_models`.

[48] B. Yee, D. Sehr, G. Dardyk, J. B. Chen, R. Muth, T. Ormandy, S. Okasaka, N. Narula, and N. Fullagar. Native Client: A Sandbox for Portable, Untrusted x86 Native Code. In *Proceedings of the 30th IEEE Symposium on Security and Privacy*, 2009.

[49] D. Yu, A. Chander, N. Islam, and I. Serikov. JavaScript Instrumentation for Browser Security. In *Proceedings of the 34th annual ACM SIGPLAN-SIGACT symposium on Principles of programming languages (POPL)*, 2007.

WebJail: Least-privilege Integration of Third-party Components in Web Mashups

Steven Van Acker, Philippe De Ryck, Lieven Desmet, Frank Piessens, Wouter Joosen
IBBT-Distrinet, Katholieke Universiteit Leuven, 3001 Leuven, Belgium
Steven.VanAcker@cs.kuleuven.be

ABSTRACT

In the last decade, the Internet landscape has transformed from a mostly static world into Web 2.0, where the use of web applications and mashups has become a daily routine for many Internet users. Web mashups are web applications that combine data and functionality from several sources or components. Ideally, these components contain benign code from trusted sources. Unfortunately, the reality is very different. Web mashup components can misbehave and perform unwanted actions on behalf of the web mashup's user.

Current mashup integration techniques either impose no restrictions on the execution of a third-party component, or simply rely on the Same-Origin Policy. A least-privilege approach, in which a mashup integrator can restrict the functionality available to each component, can not be implemented using the current integration techniques, without ownership over the component's code.

We propose WebJail, a novel client-side security architecture to enable least-privilege integration of components into a web mashup, based on high-level policies that restrict the available functionality in each individual component. The policy language was synthesized from a study and categorization of sensitive operations in the upcoming HTML 5 JavaScript APIs, and full mediation is achieved via the use of deep aspects in the browser.

We have implemented a prototype of WebJail in Mozilla Firefox 4.0, and applied it successfully to mainstream platforms such as iGoogle and Facebook. In addition, microbenchmarks registered a negligible performance penalty for page load-time (7ms), and the execution overhead in case of sensitive operations (0.1ms).

Categories and Subject Descriptors

K.6.5 [**Management of Computing and Information Systems**]: Security and Protection; H.3.5 [**Information Storage and Retrieval**]: Web-based services

Keywords

Web Application Security, Web Mashups, Sandbox, Least-privilege integration

1. INTRODUCTION

The Internet has seen an explosion of dynamic websites in the last decade, not in the least because of the power of JavaScript. With JavaScript, web developers gain the ability to execute code on the client-side, providing for a richer and more interactive web experience. The popularity of JavaScript has increased even more since the advent of Web 2.0.

Web mashups are a prime example of Web 2.0. In a web mashup, data and functionality from multiple stakeholders are combined into a new flexible and lightweight client-side application. By doing so, a mashup generates added value, which is one of the most important incentives behind building mashups. Web mashups depend on collaboration and interaction between the different mashup components, but the trustworthiness of the service providers delivering components may strongly vary.

The two most wide-spread techniques to integrate third-party components into a mashup are via script inclusion and via (sandboxed) iframe integration, as will be discussed in more detail in Section 2. The script inclusion technique implies that the third-party component executes with the same rights as the integrator, whereas the latter technique restricts the execution of the third-party component according to the Same-Origin Policy. More fine-grained techniques (such as Caja [23] or FBJS [31]) require (some form of) ownership over the code to transform or restrict the component to a known safe subset before delivery to the browser. This makes these techniques less applicable to integrate third-party components directly from their service providers.

To enable the necessary collaboration and interaction while restricting the capabilities of untrusted third-party components, web mashups should integrate components according to the least-privilege principle. This means that each of the components is only granted access to data or functionality necessary to perform its core function. Unfortunately, least-privilege integration of third-party mashup components can not be achieved with the current script-inclusion and frame-integration techniques. Moreover, the need for least-privilege integration becomes highly relevant, especially because of the augmented capabilities of the upcoming HTML5 JavaScript APIs [32] (such as access to local storage, geolocation, media capture and cross-domain communication).

In this paper, we propose WebJail, a novel client-side security architecture to enable the least-privilege integration of third-party components in web mashups. The security restrictions in place are configurable via a high-level composition policy under control of the mashup integrator, and allow the use of legacy mashup components, directly served by multiple service providers.

In summary, the contributions of this paper are:

1. a novel client-side security architecture, WebJail, that supports least-privilege composition of legacy third-party mashup-components

2. the design of a policy language for WebJail that is tuned to support the effective use of WebJail to limit access to the powerful upcoming HTML5 APIs

3. the implementation of WebJail and its policy language in Firefox, and evaluation and discussion of performance and usability

The rest of this paper is structured as follows. Section 2 sketches the necessary background, and Section 3 further elaborates the problem statement. In Section 4, the WebJail least-privilege integration architecture is presented and its three layers are discussed in more detail. Next, the prototype implementation in Firefox is described in Section 5, followed by an experimental evaluation in Section 6 and discussion in Section 7. Finally, Section 8 discusses related work, and Section 9 summarizes the contributions.

2. BACKGROUND

This section briefly summarizes the Same-Origin Policy. Next, Section 2.2 discusses how mashups are constructed and gives some insights in the state-of-practice on how third-party mashup components get integrated.

2.1 Same-Origin Policy

Currently, mashup security is based on the de facto security policy of the web: the Same-Origin Policy (SOP) [34]. An origin is a domain name-protocol-port triple, and the SOP states that scripts from one origin should not be able to access content from other origins. This prevents scripts from stealing data, cookies or login credentials from other sites. In addition to the SOP, browsers also apply a frame navigation policy, which restricts the navigation of frames to its descendants [1].

Among others, the Same-Origin Policy allows a per-origin separation of JavaScript execution contexts. Contexts are separated based on the origin of the window's document, possibly relaxed via the `document.domain` property to a right-hand, fully-qualified fragment of its current hostname. Within an execution context, the SOP does not impose any additional security restriction.

2.2 Integration of mashup components

The idea behind a web mashup is to integrate several web applications (components) and mash up their code, data and results. The result is a new web application that is more useful than the sum of its parts. Several publicly available web applications [25] provide APIs that allow them to be used as third-party components for web mashups.

To build a client-side mashup, an integrator selects the relevant in-house and third-party components, and provides the necessary glue code on an integrating web page to retrieve the third-party components from their respective service providers and let them interact and collaborate with each other.

As stated before, the two most-widespread techniques to integrate third-party components into a web mashup are through script inclusion or via (sandboxed) iframe-integration [4, 18].

Script inclusion. HTML script tags are used to execute JavaScript while a webpage is loading. This JavaScript code can be located on a different server than the webpage it is executing in. When executing, the browser will treat the code as if it originated from the same origin as the webpage itself, without any restrictions of the Same-Origin Policy.

The included code executes in the same JavaScript context, has access to the code of the integrating webpage and all of its datastructures. All sensitive JavaScript operations available to the integrating webpage are also available to the integrated component.

(Sandboxed) iframe integration. HTML iframe tags allow a web developer to include one document inside another. The integrated document is loaded in its own environment almost as if it were loaded in a separate browser window. The advantage of using an iframe in a mashup is that the integrated component from another origin is isolated from the integrating webpage via the Same-Origin Policy. However, the code running inside of the iframe still has access to all of the same sensitive JavaScript operations as the integrating webpage, albeit limited to its own execution context (i.e. origin). For instance, a third-party component can use local storage APIs, but only has access to the local storage of its own origin.

HTML 5 adds the "sandbox" attribute to the iframe element, allowing an integrator to disable all security-sensitive features through its "allow-scripts" keyword. Obviously, this very coarse-grained control has only a very limited applicability in a web mashup context.

3. PROBLEM STATEMENT

In this section, the attacker model is specified, as well as two typical attack vectors. Next, the increasing impact of insecure mashup composition is discussed in the context of the upcoming set of HTML5 specifications. Finally, the security assessment is concluded by identifying the requirements for secure mashup composition, namely the least-privilege integration of third-party mashup components.

3.1 Attacker model

Our attacker model is inspired by the definition of a *gadget attacker* in Barth *et al.* [1]. The term gadget in their definition should, in the context of this paper, be read as "third-party mashup component".

We describe the attacker in scope as follows:

Malicious third-party component provider The attacker is a malicious principal owning one or more machines on the network. The attacker is able to trick the integrator in embedding a third-party component under control of the attacker.

We assume a mashup that consists of multiple third-party components from several service providers, and an honest mashup consumer (i.e. end-user). A malicious third-party component provider attempts to steal sensitive data outside its trust boundary (e.g. reading from origin-specific client-side storage), impersonate other third-party components or the integrator (e.g. requesting access to geolocation data on behalf of the integrator) or falsely operate on behalf of the end-user towards the integrator or other service providers (e.g. requesting cross-application content with XMLHttpRequest).

We have identified two possible ways in which an attacker could present himself as a malicious third-party component provider: he could offer a malicious third-party component towards mashup integrators (e.g. via a malicious advertisement, or via a malicious clone of a popular component), or he could hack into an existing third-party component of a service provider and abuse the prior existing trust relationship between the integrator and the service provider.

In this paper, we consider the mashup integrator as trusted by the mashup consumer (i.e. end-user), and an attacker has no control over the integrator, except for the attacker's ability to embed a third-party components of his choice. In addition, we assume that the attacker has no special network abilities (such as sniffing the network traffic between client and servers), browser abilities (e.g. extension under control of the attacker or client-side malware) and is constrained in the browser by the Same-Origin Policy.

3.2 Security-sensitive JavaScript operations

The impact of running arbitrary JavaScript code in an insecure mashup composition is equivalent to acquiring XSS capabilities, either in the context of the component's origin, or in the context of the integrator. For instance, a malicious third-party component provider can invoke typical security-sensitive operations such as the retrieval of cookies, navigation of the browser to another page, launch of external requests or access and updates to the Document Object Model (DOM).

However, with the emerging HTML5 specification and APIs, the impact of injecting and executing arbitrary JavaScript has massively increased. Recently, JavaScript APIs have been proposed to access geolocation information and system information (such as CPU load and ambient sensors), to capture audio and video, to store and retrieve data from a client-side datastore, to communicate between windows as well as with remote servers.

As a result, executing arbitrary JavaScript becomes much more attractive to attackers, even if the JavaScript execution is restricted to the origin of the component, or a unique origin in case of a sandbox.

3.3 Least-privilege integration

Taking into account the attack vectors present in current mashup composition, and the increasing impact of such attacks due to newly-added browser features, there is clearly a need to limit the power of third-party mashup components under control of the attacker.

Optimally, mashup components should be integrated according to the least-privilege principle. This means that each of the components is only granted access to data or functionality necessary to perform its core function. This would enable the necessary collaboration and interaction while restricting the capabilities of untrusted third-party components.

Unfortunately, a least-privilege integration of third-party mashup components can not be achieved with the current script-inclusion and iframe-integration techniques. These techniques are too coarse-grained: either no restrictions (or only the Same-Origin Policy) are imposed on the execution of a third-party component, implicitly inviting abuse, or JavaScript is fully disabled, preventing any potential abuse but also fully killing desired functionality.

To make sure that attackers described in Section 3.1 do not exploit the insecure composition attack vectors and multiply their impact by using the security sensitive HTML5 APIs described in Section 3.2, the web platform needs a security architecture that supports least-privilege integration of web components. Since client-side mashups are composed in the browser, this architecture must necessarily be implemented in the browser. It should satisfy the following requirements:

R1 Full mediation. The security-sensitive operations need to be fully mediated. The attacker can not circumvent the security mechanisms in place.

R2 Remote component delivery. The security mechanism must allow the use of legacy third-party components and the direct delivery of components from the service provider to the browser environment.

R3 Secure composition policy. The secure composition policy must be configurable (and manageable) by the mashup integrator. The policy must allow fine-grained control over a single third-party component, with respect to the security-sensitive operations in the HTML5 APIs.

R4 Performance The security mechanism should only introduce a minimal performance penalty, unnoticeable to the end-user.

Existing technologies like e.g. Caja [23] and FBJS [31] require pre-processing of mashup components, while ConScript [21] does not work in a mashup context because it depends on the mashup component to load and enforce its own policy. A more thorough discussion of related work can be found in Section 8.

4. WEBJAIL ARCHITECTURE

To enable least-privilege integration of third-party mashup components, we propose WebJail, a novel client-side security architecture. WebJail allows a mashup integrator to apply the least-privilege principle on the individual components of the mashup, by letting the integrator express a secure composition policy and enforce the policy within the browser by building on top of the deep advice approach of ConScript [21].

The secure composition policy defines the set of security-sensitive operations that the component is allowed to invoke. Each particular operation can be allowed, disallowed, or restricted to a self-defined whitelist. Once loaded, the deep aspect layer will ensure that the policy is enforced on every accesspath to the security-sensitive operations, and that the policy can not be tampered with.

The WebJail architecture consists of three abstraction layers as shown in Figure 1. The upper layer, the *policy layer*, associates the secure composition policy with a mashup component, and triggers the underlying layers to enforce the

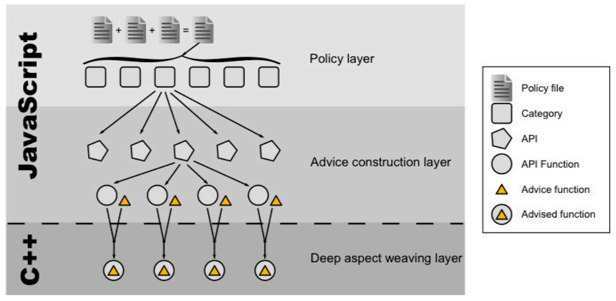


Figure 1: The WebJail architecture consists of three layers: The policy layer, the advice construction layer and the deep aspect weaving layer.

policy for the given component. The lower layer, the *deep aspect weaving layer*, enables the deep aspect support with the browser's JavaScript engine. The *advice construction layer* in between takes care of mapping the higher-level policy blocks onto the low-level security-sensitive operations via a 2-step policy refinement process.

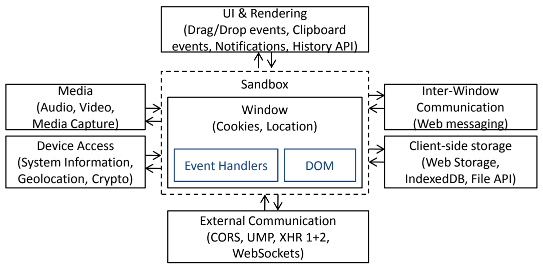
In this section, the three layers of the WebJail will be described in more detail. Next, Section 5 will discuss a prototype implementation of this architecture in Mozilla Firefox.

4.1 Policy layer

The policy layer associates the secure composition policy with the respective mashup component. In this section, an analysis of security-sensitive operations in the HTML5 APIs is reported and discussed, as well as the secure composition policy itself.

4.1.1 Security-sensitive JavaScript operations

As part of this research, we have analyzed the emerging specifications and browser implementations, and have identified 86 security-sensitive operations, accessible via JavaScript APIs. We have synthesized the newly-added features of these specifications in Figure 2, and we will briefly summarize each of the components in the next paragraphs. Most of these features rely on (some form of) user-consent and/or have origin-restrictions in place.


UI & Rendering
(Drag/Drop events, Clipboard events, Notifications, History API)
Sandbox
Window
(Cookies, Location)
Event Handlers
DOM
Media
(Audio, Video, Media Capture)
Device Access
(System Information, Geolocation, Crypto)
Inter-Window Communication
(Web messaging)
Client-side storage
(Web Storage, IndexedDB, File API)
External Communication
(CORS, UMP, XHR 1+2, WebSockets)
Figure 2: Synthesized model of the emerging HTML5 APIs

Central in the model is the *window* concept, containing the document. The window manifest itself as a browser window, a tab, a popup or a frame, and provides access to the location and history, event handlers, the document and its associated DOM tree. Event handlers allow to register for a specific event (e.g. being notified of mouse clicks), and access to the DOM enables a script to read or modify the document's structure on the fly. Additionally, a *sandbox* can impose coarse-grained restrictions on an iframe, as mentioned in Section 2.2.

Inter-frame communication allows sending messages between windows (e.g. between mashup components). This includes window navigation, as well as Web Messaging (postMessage).

Client-side storage enables applications to temporarily or persistently store data. This can be achieved via Web Storage, IndexedDB or the File API.

External communication features such as CORS, UMP, XMLHttpRequest level 1 and 2, and websockets allow an application to communicate with remote websites, even in cross-origin settings.

Device access allows the web application to retrieve contextual data (e.g. geolocation) as well as system information such as battery level, CPU information and ambient sensors.

Media features enable a web application to play audio and video fragments, as well as capture audio and video via a microphone or webcam.

The UI and rendering features allow subscription to clipboard and drag-and-drop events, issuing desktop notifications and populating the history via the History API.

For a more thorough analysis of the HTML5 APIs, we would like to refer to an extensive security analysis we have carried out, commissioned by the European Network and Information Security Agency (ENISA) [7].

4.1.2 Secure composition policy

The policy layer associates the secure composition policy with a mashup component, and deploys the necessary security controls via the underlying layers. As composition granularity, we have chosen the iframe level; i.e. mashup components are each loaded in their separate iframe.

In particular, within WebJail the secure composition policy is expressed by the mashup integrator, and attached to a particular component via a newly-introduced *policy* attribute of the iframe element of the component to be loaded.

```
<iframe src="http://untrusted.com/compX/"
     policy="https://integrator.com/compX.policy"/>
```

We have grouped the identified security-sensitive operations in the HTML5 APIs in nine disjoint categories, based on their functionality: DOM access, Cookies, External communication, Inter-frame communication, Client-side storage, UI & Rendering, Media, Geolocation and Device access.

For a third-party component, each category can be fully disabled, fully enabled, or enabled only for a self-defined whitelist. The whitelists contain category-specific entries. For example, a whitelist for the category "DOM Access" contains the ids of the elements that might be read from or updated in the DOM. The nine security-sensitive categories are listed in Table 1, together with their underlying APIs, the amount of security-sensitive functions in each API, and their WebJail whitelist types.

The secure composition policy expresses the restrictions for each of the security-sensitive categories, and an example policy is shown below. Unspecified categories are disallowed by default, making the last line in the example policy obsolete.

```
{ "framecomm" : "yes",
  "extcomm" : [ "google.com", "youtube.com" ],
  "device" : "no" }
```

Categories and APIs (# op.)	Whitelist
DOM Access	ElemReadSet, ElemWriteSet
DOM Core (17)	
Cookies	KeyReadSet, KeyWriteSet
cookies (2)	
External Communication	DestinationDomainSet
XHR, CORS, UMP (4)	
WebSockets (5)	
Server-sent events (2)	
Inter-frame Communication	DestinationDomainSet
Web Messaging (3)	
Client-side Storage	KeyReadSet, KeyWriteSet
Web Storage (5)	
IndexedDB (16)	
File API (4)	
File API: Dir. and Syst. (11)	
File API: Writer (3)	
UI and Rendering	
History API (4)	
Drag/Drop events (3)	
Media	
Media Capture API (3)	
Geolocation	
Geolocation API (2)	
Device Access	SensorReadSet
System Information API (2)	
Total number of security-sensitive operations: 86	

Table 1: Overview of the sensitive JavaScript operations from the HTML 5 APIs, divided in categories.

It is important to note that WebJails or regular frames can be used inside WebJails. In such a case, the functionality in the inner frame is determined by the policies imposed on enclosing frames, in addition to its own policy (if it has one, as is the case with a WebJail frame). Allowing sensible cascading of policies implies that "deeper" policies can only make the total policy more strict. If this were not the case, a WebJail with a less strict policy could be used to "break out" of the WebJail restrictions.

The semantics of a policy entry for a specific category can be thought of as a set. Let $\mathcal{V}$ be the set of all possible values that can be listed in a whitelist. The "allow all" policy would then be represented by the set $\mathcal{V}$ itself, a whitelist would be represented by a subset $w \subseteq \mathcal{V}$ and the "allow none" policy by the empty set ϕ. The relationship "x is at least as strict as y" can be represented as $x \subseteq y$. Using this notation, the combined policy p of 2 policies a and b is the intersection $p = a \cap b$, since $p \subseteq a$ and $p \subseteq b$.

After loading, parsing and combining all the policies applicable to the WebJail protected iframe, the policy is enforced via the underlying layers.

4.2 Advice construction layer

The task of the advice construction layer is to build advice functions based on the high-level policy received from the policy layer, and apply these advice functions on the low-level security-sensitive operations via deep aspect technology in the deep advice weaving layer.

To do so, the advice construction layer applies a 2-step refinement process. For each category of the secure composition policy, the set of relevant APIs is selected. Next for each API, the individual security-sensitive operations are processed. Consider for instance that a whitelist of type "KeyReadSet"[1] is specified for the client-side storage in the composition policy. This is first mapped to the various storage APIs in place (such as Web Storage and File API), and then advice is constructed for the security-sensitive operations in the API (e.g. for accessing the *localStorage* object).

The advice function decides, based on the policy, whether or not the associated API function will be called: if the policy for the API function is "allow all", or "allow some" and the whitelist matches, then the advice function allows the call. Otherwise, the call is blocked.

On successful completion of its job, the advice construction layer has advice functions for all the security-sensitive operations across the nine categories relevant for the specific policy. Next, the advices are applied on the original operations via the deep advice weaving layer.

4.3 Deep aspect weaving layer

The *(advice, operation)* pairs received from the advice construction layer are registered into the JavaScript engine as deep advice. The result of this weaving is that the original API function is replaced with the advice function, and that all accesspaths to the API function now go through the advice function. The advice function itself is the only place where a reference to the original API function exists, allowing it to make use of the original functionality when desired.

5. PROTOTYPE IMPLEMENTATION

To show the feasibility and test the effectiveness of WebJail, we implemented a prototype by modifying Mozilla Firefox 4.0b10pre.The modifications to the Mozilla code are localized and consist of ±800 lines of new code (±300 JavaScript, ±500 C++), spread over 3 main files. The prototype currently supports the security-sensitive categories external and inter-frame communication, client-side storage, UI and rendering (except for drag/drop events) and geolocation.

Each of the three layers of the implementation will be discussed now in more detail.

5.1 Policy layer

The processing of the secure composition policy via the *policy* attribute happens in the frame loader, which handles construction of and loading content into frames. The specified policy URL is registered as the policy URL for the frame to be loaded, and any content loaded into this frame will be subject to that WebJail policy, even if that content issues a refresh, submits a form or navigates to another URL.

When an iframe is enclosed in another iframe, and both specify a policy, the combinatory rules defined in Section 4 are applied on a per-category basis. To ease up parsing of a policy file, we have chosen to use the JavaScript Object Notation (JSON).

Once the combined policy for each category has been calculated, the list of APIs in that category is passed to the advice construction layer, along with the combined policy.

5.2 Advice construction layer

The advice construction layer builds advice functions for individual API functions. For each API, the advice construction layer knows what functions are essential to enforce the policy and builds a specific advice function that enforces it.

The advice function is a function that will be called instead of the real function. It will determine whether or not the real function will be called based on the policy and the arguments passed in the function call. Advice functions in WebJail are written in JavaScript and should expect 3 ar-

[1]Such a whitelist contains a set of keys that may be read

guments: a function object that can be used to access the original function, the object on which the function was invoked (i.e. the `this` object) and a list with the arguments passed to the function.

```
1  function makeAdvice(whitelist) {
2    var myWhitelist = whitelist;
3
4    return function(origf, obj, vp) {
5      if(myWhitelist.ROindexOf(vp[0])>=0) {
6        return origf.ROapply(obj, vp);
7      } else {
8        return false;
9      }
10   };
11 }
12
13 myAdvice = makeAdvice(['foo', 'bar']);
14 registerAdvice(myFunction, myAdvice);
15 disableAdviceRegistration();
```

Figure 3: Example advice function construction and weaving

The construction of a rather generic example advice function is shown in Figure 3. The listing shows a function `makeAdvice`, which returns an advice function as a closure containing the whitelist. Whenever the advice function is called for a function to which the first argument (`vp[0]`) is either 'foo' or 'bar', then the original function is executed. Otherwise, the advice function returns false.

Note that in the example, `ROindexOf` and `ROapply` are used. These functions were introduced to prevent prototype poisoning attacks against the WebJail infrastructure. They provide the same functionality as `indexOf` and `apply`, except that they have the `JSPROP_READONLY` and `JSPROP_PERMANENT` attributes set so they can not be modified or deleted.

Next, each *(advice, operation)* pair is passed on to the deep aspect weaving layer to achieve the deep aspect weaving.

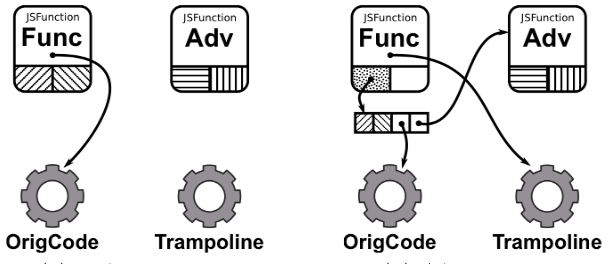
5.3 Deep aspect weaving layer

The deep aspect weaving layer makes sure that all code-paths to an advised function pass through its advice function. Although the code from WebJail is the first code to run in a WebJail iframe, we consider the scenario that there can be code or objects in place that already reference the function to be advised. It is necessary to maintain the existing references to a function, if they exist, so that advice weaving does not break code unintentionally.

The implementation of the deep aspect weaving layer is inspired by ConScript. To register deep advice, we introduce a new function called `registerAdvice`, which takes 2 arguments: the function to advise (also referred to as the 'original' function) and its advice function. Line 14 of Figure 3 illustrates the usage of the `registerAdvice` function.

In Spidermonkey, Mozilla's JavaScript engine, all JavaScript functions are represented by `JSFunction` objects. A `JSFunction` object can represent both a native function, as well as a JIT compiled JavaScript function. Because WebJail enforces policies on JavaScript APIs and all of these are implemented with native functions, our implementation only considers `JSFunction` objects which point to native code[2].

The process of registering advice for a function is schematically illustrated in Figure 4. Consider a native function `Func` and its advice function `Adv`. Before deep aspect weaving, the `JSFunction` object of `Func` contains a reference to a native C++ function `OrigCode`.



(a) Before weaving (b) After weaving

Figure 4: Schematic view of deep aspect weaving.

At weaving time, the value of the function pointer in `Func` (which points to `OrigCode`) and a reference to `Adv` are backed up inside the `Func` object. The function pointer inside `Func` is then directed towards the `Trampoline` function, which is an internal native C++ function provided by WebJail.

At function invocation time, the `Trampoline` function will be called as if it were the original function (`OrigCode`). This function can retrieve the values backed up in the weaving phase. From the backed up function pointer pointing to `OrigCode`, a new anonymous `JSFunction` object is created. This anonymous function, together with the current `this` object and the arguments to the `Trampoline` function are passed to the advice function `Adv`. Finally, the result from the advice function is returned to the calling code.

In reality, the `registerAdvice` function is slightly more complicated. In each `JSFunction` object, SpiderMonkey allocates 2 private values, known as "reserved slots", which can be used by Firefox to store opaque data. As shown in Figure 4, the reserved slots of `Func` (hatched diagonally) are backed up in the weaving phase together with the other values. During invocation time, these reserved slots are then restored into the anonymous function mentioned earlier.

Note that all code that referenced `Func` still works, although calls to this function will now pass through the advice function `Adv` first. Also note that no reference to the original code `OrigCode` is available. The only way to call this code is by making use of the advice function.

To prevent any other JavaScript code from having access to the `registerAdvice` function, it is disabled after all advice from the policy has been applied. For this purpose, WebJail provides the `disableAdviceRegistration` function, which disables the use of the `registerAdvice` function in the current JavaScript context.

6. EVALUATION

6.1 Performance

We performed micro-benchmarks on WebJail to evaluate its performance overhead with regard to page load-time and function execution. The prototype implementation is built on Mozilla Firefox 4.0b10pre, and compiled with the GNU C++ compiler v4.4.4-14ubuntu5. The benchmarks were performed on an Apple MacBook Pro 4.1, with an Intel Core 2 Duo T8300 CPU running at 2.40GHz and 4GB of memory, running Ubuntu 10.10 with Linux kernel version `2.6.35-28-generic`.

[2]Although WebJail could be implemented for non-native functions as well.

6.1.1 *Page load-time overhead*

To measure the page load-time overhead, we created a local webpage (`main.html`) that embeds another local page (`inner.html`) in an iframe with and without a local policy file. `inner.html` records a timestamp (`new Date().getTime()`) when the page starts and stops loading (using the body `onload` event). WebJail was modified to record the starttime before anything else executes, so that policy retrieval, loading and application is taken into account. After the results are submitted, `main.html` reloads.

We averaged the results of 1000 page reloads. Without WebJail, the average load-time was 16.22ms ($\sigma = 3.74$ms). With WebJail, the average is 23.11ms ($\sigma = 2.76$ms).

6.1.2 *Function execution overhead*

Similarly, we used 2 local pages (`main.html` and `inner.html`) to measure function execution overhead. `inner.html` measures how long it takes for 10000 iterations of a piece of code to execute. We measured 2 scenarios: a typical XMLHttpRequest invocation (constructor, `open` and `send` functions) and a localStorage set and get (`setItem` and `getItem`). Besides measuring a baseline without WebJail policy, we measured each scenario when restricted by 3 different policies: "allow all", "allow none" and a whitelist with 5 values. The averages are summarized in Table 2.

	XMLHttpRequest	localStorage
Baseline	1.25 ms	0.37 ms
"Allow all"	1.25 ms (+ 0%)	0.37 ms (+ 0%)
"Allow none"	0.07 ms (- 94.4%)	0.04 ms (- 89.2 %)
Whitelist	1.33 ms (+ 6.4%)	0.47 ms (+ 27%)

Table 2: Function execution overhead

To conclude, we have registered a negligible performance penalty for our WebJail prototype: a page load-time of 7ms, and an execution overhead in case of sensitive operations about 0.1ms.

6.2 Security

As discussed in Subsection 5.3, the `registerAdvice` function disconnects an available function and makes it available only to the advice function. Because of the use of deep aspects, we can ensure that no other references to the original function are available in the JavaScript environment, even if such references already existed before `registerAdvice` was called. We have successfully verified this full mediation of the deep aspects using our prototype implementation.

Because advice functions are written in JavaScript and the advice function has the only reference to the original function, it would be tempting for an attacker to attack the WebJail infrastructure. The retrieval and application of a WebJail policy happens before any other code is executed in the JavaScript context. In addition, the `registerAdvice` function is disabled once the policy has been applied. The only remaining attack surface is the advice function during its execution. The advice functions constructed by the advice construction layer are functionally equivalent to the example advice function created in Figure 3. We know of 3 attack vectors: prototype poisoning of `Array.prototype.indexOf` and `Function.prototype.apply`, and `toString` redefinition on `vp[0]` (the first argument to the example advice function in Figure 3). By introducing the readonly copies `ROindexOf` and `ROapply` (See Subsection 5.2), we prevent an attacker from exploiting the first 2 attack vectors. The third vector, `toString` redefinition, was verified in our prototype implementation and is not an issue because `toString` is never called on the argument `vp[0]`.

6.3 Applicability

To test the applicability of the WebJail architecture, we have applied our prototype implementation to mainstream mashup platforms, including iGoogle and Facebook. As part of the setup, we have instrumented responses from these platforms to include secure composition policies, by automatically injecting a *policy* attribute in selected iframes. Next, we have applied both permissive composition policies as well as restricted composition policies and verified that security-sensitive operations for the third-party components were executed as usual in the first case, and blocked in the latter case. For instance, as part of the applicability tests, we applied WebJail to control Geolocation functionality in the Google Latitude[11] component integrated into iGoogle, as well as external communication functionality of the third-party Facebook application "Tweets To Pages"[14] integrated into our Facebook page.

7. DISCUSSION AND FUTURE WORK

In the previous sections, we have showed the feasibility of the WebJail architecture via a prototype implementation in Firefox, and evaluated the performance, security and applicability. By applying micro-benchmarks, we measured a negligible overhead, we discussed how the WebJail architecture achieves full mediation via deep aspect weaving, and we briefly illustrated the applicability of WebJail in mainstream mashup platforms.

In this section, we will discuss some points of attention in realizing least-privilege integration in web mashups and some opportunities for further improvements.

First, the granularity chosen for the secure composition policies for WebJail is primarily driven by the ease of configuration for the mashup integrator. We strongly believe that the category level of granularity increases the adoption potential by integrators and browsers, for instance compared to semantically rich and expressive security policies as is currently the case in wrapper approaches or ConScript. In fact, we chose to introduce this policy abstraction to let the integrator focus on the "what" rather than the "how". A next step could be to define policy templates per mashup component type (e.g. advertisement and geotagging components).

Nevertheless, more fine-grained policies could also be applied to achieve least-privilege integration, but one should be aware of the potential risk of creating an inverse sandbox. The goal of a least-privilege integration architecture, such as WebJail, is to limit the functionality available to a (possibly) malicious component. In case the policy language is too expressive, an attacker could use this technology to achieve the inverse. An attacker could integrate a legitimate component into his website and impose a malicious policy on it. The result is effectively a hardcoded XSS attack in the browser. For instance, the attacker could introduce an advice that leaks all sensitive information out of a legitimate component as part of its least-privilege composition policy without being stopped by the Same-Origin Policy.

One particular area where we see opportunities for more fine-grained enforcement are cross-domain interactions. Ongoing research on Cross-Site Request Forgery (CSRF) [5, 6, 28, 20] already differentiates between benign and potentially malicious cross-domain requests, and restricts the latter class as part of a browser extension. This line of research could be seen as complementary to the presented approach, and a combination of both would allow a more fine-grained enforcement for cross-domain interactions.

Second, a possible technique to escape a modified JavaScript execution context in an iframe, would be to open a new window and execute JavaScript in there. We have anticipated this attack by hardcoding policies for e.g. the `window.open` function. This is however not the best approach. The upcoming HTML 5 specs include the sandbox attribute for iframes. This specification states that a sandbox should prevent content from creating new auxiliary browsing contexts. Mozilla Firefox does not support the sandbox attribute yet. The hardcoded policy for `window.open` is a quick fix while we are working on our own full implementation of the sandbox attribute in Mozilla Firefox.

Another way to escape WebJail is to access the window object of the parent or a sibling frame and make use of the functions in that JavaScript context (e.g. `parent.navigator.geolocation.getCurrentPosition`). In such a scenario, accessing another JavaScript context falls under the Same-Origin Policy and will only be possible if both the caller and callee are in the same origin. To avoid this attack, the WebJail implementation must restrict access to sensitive operations in other execution contexts under the Same-Origin Policy.

Thirdly, the categories in the policy files of WebJail are a result of a study of the sensitive JavaScript operations in the new HTML5 APIs. Most of the HTML5 APIs are working drafts and might change in the future. The category list in WebJail is therefore an up-to-date snapshot, but might be subject to change in the future. Even after the specifications for HTML5 are officially released, the functionality in browsers might keep changing. To cope with this evolving landscape, WebJail can easily be extended to support additional categories and APIs as well.

Finally, the WebJail architecture is tailored to support least-privilege integration in mashups that are built via iframe-integration. An interesting future track is to investigate how to enable browsers to support least-privilege script-inclusion integration as well. Since in such a scenario, one can not build on the fact that a separate execution context is created, we expect this to be a challenging trajectory.

8. RELATED WORK

There is a broad set of related work that focuses on the integration of untrusted JavaScript code in web applications.

JavaScript subsets.

A common technique to prevent undesired behavior is to restrict the untrusted code (i.e. the third-party component) to a safe subset of JavaScript. The allowed operations within the subset prevent the untrusted code from obtaining elevated privileges, unless explicitly allowed by the integrator.

ADSafe[3] and FBJS[31] requires third-party components to be written in a JavaScript subset that is known to be safe. The ADSafe subset removes several unsafe features from JavaScript (e.g. global variables, eval, ...) and provides safe alternatives through the `ADSAFE` object. Caja[23], Jacaranda[15] and Live Labs' Websandbox[22] take a different approach. Instead of heavily restricting the developer's language, they transform the JavaScript code into a safe version. The transformation process is based on both static analysis and rewriting to integrate runtime checks.

These techniques effectively support client-side least-privilege integration of mashup components. The main disadvantage is the tight coupling of the security features with the third-party component code. This requires control over the code, either at development or deployment time, which conflicts with legacy components and remote component delivery (R2), and reduces the applicability to mashup scenarios where the integrator delivers the components to the browser.

JavaScript instrumentation and access mediation.

Instead of restricting a third-party component to a JavaScript subset, access to specific security-sensitive operations can be mediated. Mediation can consist of blocking the call, or letting a policy decide whether or not to allow it.

BrowserShield[26] is a server-side rewriting technique, that rewrites certain JavaScript functions to use safe equivalents. These safe equivalents are implemented in the "bshield" object that is introduced through the BrowserShield JavaScript libraries that are injected into each page. BrowserShield makes use of a proxy to inject its code into a webpage.

Self-protecting JavaScript[24, 19] is a client-side wrapping technique that applies advice around JavaScript functions, without requiring any browser modifications. The wrapping code and advice are provided by the server and are executed first, ensuring a clean environment to start from. The advice is *non-deep* advice, meaning that by protecting one operation, different access paths to the same operation are not automatically protected. The main challenge of this approach is to ensure full mediation (R1) without breaking the component's legitimate functionality (e.g. via removal of prototypes), since both policy and third-party component code live in the same JavaScript context.

Browser-Enforced Embedded Policies (BEEP)[16] injects a policy script at the server-side. The browser will call this policy script before loading another script, giving the policy the opportunity to vet the script about to be loaded. The loading process will only continue after the approval of the policy. This approach offers control over which scripts are loaded, but is too coarse grained to assign privileges to specific components.

ConScript[21] allows the enforcement of fine-grained security policies for JavaScript in the browser. The approach is similar to self-protecting JavaScript, except that ConScript uses *deep* advice, thus protects all access paths to a function. The price for using deep advice is the need for client-side support in the JavaScript engine. A limitation of ConScript is that policies are not composition policies: the policies are provided by and applied to the same webpage, which conflicts with remote component delivery (R2) and secure composition policy configurable by the integrator (R3).

In contrast to the techniques described above, WebJail offers the integrator the possibility to define a policy that restricts the behavior of a third-party component in an isolated way. Additionally, all of the techniques above use JavaScript as a policy language. This amount of freedom complicates the writing of secure policies: protection against all the emerging HTML5 APIs is fully up to policy writer and can be error-prone, a problem that the WebJail policy language is not susceptible to.

Web application code and data analysis.

A common protection technique against XSS vulnerabilities or attacks is server-side code or data analysis. Even though these techniques can only be used to check if a component matches certain security requirements and do not enforce a policy, we still discuss them here, since they are a server-side way to ensure that a component meets certain least-privilege integration requirements *out-of-the-box*.

Gatekeeper[12] is a mostly static [sic] enforcement mechanism designed to defend against possibly malicious JavaScript widgets on a hosting page. Gatekeeper analyzes the complete JavaScript code together with the hosting page. In addition, Gatekeeper uses runtime enforcement to disable dynamic JavaScript features.

XSS-Guard[2] aims to detect and remove scripts that are not intended to be present in a web application's output, thus effectively mitigating XSS attacks. XSS-Guard dynamically learns what set of scripts is used for an HTTP request. Using this knowledge, subsequent requests can be protected.

Recently, Mozilla proposed the Content Security Policy (CSP) [29], which allows the integrator to insert a security policy via response headers or meta tags. Unfortunately, CSP only supports restrictions on a subset of the security-sensitive operations discussed in this paper, namely operations potentially leading to content injection (e.g. script inclusion and XHR).

Information flow control.

Information flow control techniques can be used to detect unauthorized information sharing or leaking between origins or external parties. This is extremely useful for applications that are allowed to use sensitive data, such as a location, but are not allowed to share that data.

Both Magazinius et al.[18] and Li et al.[17] have proposed an information flow control technique that prevents unauthorized sharing of data. Additionally, both techniques support authorized sharing by means of declassification, where a certain piece of data is no longer considered sensitive.

Secure multi-execution[9] detects information leakage by simultaneously running the code for each security level. This approach is a robust way to detect information leakage, but does not support declassification.

Information flow control techniques themselves are not suited for enforcing least-privilege integration. Likewise, WebJail is not suited to enforce information flow control, since it would be difficult to cover all possible leaks. Both techniques are complementary and can be used together to ensure least-privilege integration without unauthorized information leaking.

Isolating content using specialized HTML.

Another approach to least-privilege integration is the isolation of untrusted content. By explicitly separating the untrusted code, it becomes easier to restrict its behavior, for example by preventing script execution.

The "untrusted" attribute[10] on a div element aims to allow the browser to make the difference between trusted and untrusted code. The idea is to enclose any untrusted content with such a div construct. This technique fails to defend against injecting closing tags, which would trivially circumvent the countermeasure.

The new "sandbox" attribute of the iframe element in HTML 5[13] provides a safer alternative, but is very coarse-grained. It only supports limited restrictions, and as far as JavaScript APIs are concerned, it only supports to completely enable or disable JavaScript.

ADJail[30] is geared towards securely isolating ads from a hosting page for confidentiality and integrity purposes, while maintaining usability. The ad is loaded on a shadow page that contains only those elements of the hosting page that the web developer wishes the ad to have access to. Changes to the shadow page are replicated to the hosting page if those changes conform to the specified policy. Likewise, user actions on the hosting page are mimicked to the shadow page if allowed by the policy. ADJail limits DOM access and UI interaction with the component, but does not restrict the use of all other sensitive operations like WebJail can.

User-provided policies.

Mozilla offers *Configurable Security Policies*[27], a user-configurable policy that is part of the browser. The policy allows the user to explicitly enable or disable certain capabilities for specific internet sites. An example is the option to disallow a certain site to open a popup window. Some parts of this idea have also been implemented in the Security zones of Internet Explorer.

The policies and enforcement mechanism offered by this technique resemble WebJail. The major difference is that these policies are user-configurable, and thus not under control of the integrator. Additionally, the policies do not support a different set of rules for the same included content, in two different scenarios, whereas WebJail does.

9. CONCLUSION

In this paper we have presented WebJail, a novel client-side security architecture to enables least-privilege integration of third-party components in web mashups. The WebJail security architecture is compatible with legacy mashup components, and allows the direct delivery of components from the service providers to the browser.

We have designed a secure composition language for WebJail, based on a study of security-sensitive operations in HTML5 APIs, and achieved full mediation by applying deep aspect weaving within the browser.

We have implemented a prototype of WebJail in Mozilla Firefox 4.0, and applied it successfully to mainstream platforms such as iGoogle and Facebook. In addition, we have evaluated the performance of the WebJail implementation using micro-benchmarks, showing that both the page load-time overhead (±7ms) and the execution overhead of a function advised with a whitelist policy (±0.1ms) are negligible.

10. ACKNOWLEDGMENTS

This research is partially funded by the Interuniversity Attraction Poles Programme Belgian State, Belgian Science Policy, IBBT, IWT, the Research Fund K.U.Leuven and the EU-funded FP7-projects WebSand and NESSoS.

The authors would also like to thank Maarten Decat and Willem De Groef for their contribution to early proof-of-concept implementations [8, 33] to test the feasibility of the presented research.

11. REFERENCES

[1] A. Barth, C. Jackson, and J. C. Mitchell. Securing frame communication in browsers. *Commun. ACM*,

52:83–91, June 2009.

[2] P. Bisht and V. Venkatakrishnan. Xss-guard: Precise dynamic prevention of cross-site scripting attacks. In *5th GI International Conference on Detection of Intrusions & Malware, and Vulnerability Assesment*, July 2008.

[3] D. Crockford. ADsafe – making JavaScript safe for advertising. `http://adsafe.org/`.

[4] P. De Ryck, M. Decat, L. Desmet, F. Piessens, and W. Joosen. Security of web mashups: a survey. In *15th Nordic Conference in Secure IT Systems (NordSec 2010)*. Springer, 2011.

[5] P. De Ryck, L. Desmet, T. Heyman, F. Piessens, and W. Joosen. Csfire: Transparent client-side mitigation of malicious cross-domain requests. In *Lecture Notes in Computer Science*, volume 5965, pages 18–34. Springer Berlin / Heidelberg, February 2010.

[6] P. De Ryck, L. Desmet, W. Joosen, and F. Piessens. Automatic and precise client-side protection against csrf attacks. In V. Atluri and C. Diaz, editors, *Computer Security - ESORICS 2011*, volume 6879 of *Lecture Notes in Computer Science*, pages 100–116. Springer Berlin / Heidelberg, 2011.

[7] P. De Ryck, L. Desmet, P. Philippaerts, and F. Piessens. A security analysis of next generation web standards. Technical report, G. Hogben and M. Dekker (Eds.), European Network and Information Security Agency (ENISA), July 2011.

[8] M. Decat. Ondersteuning voor veilige Web Mashups. Master's thesis, Katholieke Universiteit Leuven, 2010.

[9] D. Devriese and F. Piessens. Noninterference through Secure Multi-execution. *2010 IEEE Symposium on Security and Privacy*, pages 109–124, 2010.

[10] A. Felt, P. Hooimeijer, D. Evans, and W. Weimer. Talking to strangers without taking their candy: isolating proxied content. In *SocialNets '08: Proceedings of the 1st Workshop on Social Network Systems*, pages 25–30, New York, NY, USA, 2008. ACM.

[11] Google. Google Latitude. `https://www.google.com/latitude/`.

[12] S. Guarnieri and B. Livshits. Gatekeeper: Mostly static enforcement of security and reliability policies for javascript code. In *Proceedings of the Usenix Security Symposium*, Aug. 2009.

[13] I. Hickson and D. Hyatt. HTML 5 Working Draft - The sandbox Attribute. `http://www.w3.org/TR/html5/the-iframe-element.html#attr-iframe-sandbox`, June 2010.

[14] Involver. Tweets To Pages. `http://www.facebook.com/TweetsApp`.

[15] Jacaranda. Jacaranda. `http://jacaranda.org`.

[16] T. Jim, N. Swamy, and M. Hicks. Defeating Script Injection Attacks with Browser-Enforced Embedded Policies. In *WWW '07: Proceedings of the 16th international conference on World Wide Web*, pages 601–610, New York, NY, USA, 2007. ACM.

[17] Z. Li, K. Zhang, and X. Wang. Mash-if: Practical information-flow control within client-side mashups. In *Dependable Systems and Networks (DSN), 2010 IEEE/IFIP International Conference on*, pages 251 –260, 28 2010-july 1 2010.

[18] J. Magazinius, A. Askarov, and A. Sabelfeld. A lattice-based approach to mashup security. In *Proceedings of the 5th ACM Symposium on Information, Computer and Communications Security*, ASIACCS '10, pages 15–23, New York, NY, USA, 2010. ACM.

[19] J. Magazinius, P. Phung, and D. Sands. Safe wrappers and sane policies for self protecting javascript. In *The 15th Nordic Conf. in Secure IT Systems. Springer Verlag*, 2010.

[20] G. Maone. Noscript 2.0.9.9. `http://noscript.net/`, 2011.

[21] L. Meyerovich and B. Livshits. ConScript: Specifying and enforcing fine-grained security policies for Javascript in the browser. In *IEEE Symposium on Security and Privacy*, May 2010.

[22] Microsoft Live Labs. Live Labs Websandbox. `http://websandbox.org`.

[23] M. S. Miller, M. Samuel, B. Laurie, I. Awad, and M. Stay. Caja - safe active content in sanitized JavaScript. Technical report, Google Inc., June 2008.

[24] P. H. Phung, D. Sands, and A. Chudnov. Lightweight self-protecting javascript. In *Proceedings of the 4th International Symposium on Information, Computer, and Communications Security*, ASIACCS '09, pages 47–60, New York, NY, USA, 2009. ACM.

[25] Programmable Web. Keeping you up to date with APIs, mashups and the Web as platform. `http://www.programmableweb.com/`.

[26] C. Reis, J. Dunagan, H. J. Wang, O. Dubrovsky, and S. Esmeir. BrowserShield: vulnerability-driven filtering of dynamic HTML. In *OSDI '06: Proceedings of the 7th symposium on Operating systems design and implementation*, pages 61–74, Berkeley, CA, USA, 2006. USENIX Association.

[27] J. Ruderman. Configurable Security Policies. `http://www.mozilla.org/projects/security/components/ConfigPolicy.html`.

[28] J. Samuel. Requestpolicy 0.5.20. `http://www.requestpolicy.com`, 2011.

[29] S. Stamm, B. Sterne, and G. Markham. Reining in the web with content security policy. In *Proceedings of the 19th international conference on World wide web*, WWW '10, pages 921–930, New York, NY, USA, 2010. ACM.

[30] M. Ter Louw, K. T. Ganesh, and V. Venkatakrishnan. Adjail: Practical enforcement of confidentiality and integrity policies on web advertisements. In *19th USENIX Security Symposium*, Aug. 2010.

[31] The FaceBook Team. FBJS. `http://wiki.developers.facebook.com/index.php/FBJS`.

[32] W3C. W3C Standards and drafts - Javascript APIs. `http://www.w3.org/TR/#tr_Javascript_APIs`.

[33] Willem De Groef. ConScript For Firefox. `http://cqrit.be/conscript/`.

[34] M. Zalewski. Browser security handbook. `http://code.google.com/p/browsersec/wiki/Main`, 2010.

Key Escrow from a Safe Distance

Looking Back at the Clipper Chip

Matt Blaze
University of Pennsylvania
blaze@cis.upenn.edu

ABSTRACT

In 1993, the US Government proposed a novel (and highly controversial) approach to cryptography, called *key escrow*. Key escrow cryptosystems used standard symmetric- and public- key ciphers, key management techniques and protocols, but with one added feature: a copy of the current session key, itself encrypted with a key known to the government, was sent at the beginning of every encrypted communication stream. In this way, if a government wiretapper encountered ciphertext produced under a key escrowed cryptosystem, recovering the plaintext would be a simple matter of decrypting the session key with the government's key, regardless of the strength of the underlying cipher algorithms. Key escrow was intended to strike a "balance" between the needs for effective communications security against *bad guys* on the one hand and the occasional need for the *good guys* to be able to recover meaningful content from (presumably) legally-authorized wiretaps.

It didn't quite work out that way.

1. CARROTS, STICKS & ENCRYPTION

The 1990's were interesting times for cryptography. The civilian academic and commercial worlds were becoming seriously interested in this previously obscure and arcane subject, bolstered by new and exciting cryptologic advances coupled with a brave new technological landscape in which securing information was understood to be something that would soon become a very important problem. Information technology was getting inexorably faster, cheaper and better, with the notable exception of security, which seemed actually to get *worse* with every iteration of Moore's law. Cryptography, we believed (or hoped), could come to the rescue, delivering its promise of securing information carried over the increasingly insecure media of the Internet and the "information superhighways" spoken about by visionaries of the time. Cryptography, we increasingly sensed, would soon no longer be merely the esoteric stuff of spies, armies, and governments, but would become an integral part of the public information economy.

But there was a hitch.

Securing information, the government reminded us, can be a double-edged sword. In the classic model for encrypted communication, upstanding citizens *Alice* and *Bob* seek to cryptographically protect their communication from *Eve*, an evildoing eavesdropper. But in a world where transparent cryptography is built in to every telephone, computer and network node, giving the honest Alices and Bobs of the world a mechanism to keep their secrets from Eve might also give the less savory the ability to evade legal surveillance. In other words, what if Eve is occasionally the *good guy?*

And so even before the Web became synonymous with the Internet, before a single bit of encrypted SSL traffic was generated, lines were being drawn for what would become an epic battle that would preoccupy a generation of cryptographers. (And it was a bad time for that community to be preoccupied; this was the same time that the basic foundations of the of web and other critical communications technologies were designed and put into place. We've been living with the security, or lack of security, built in to that infrastructure ever since).

And the eavesdroppers had some leverage. While there were no laws in the United States preventing Alice and Bob from encrypting their communications as unbreakably as they wanted, things weren't so simple for the manufacturers of the hardware and software they might need to effectively do it. Encryption technology, it turned out, was classified as a munition, regulated under the same laws that control trafficking in rocket launchers and nuclear weapons. These laws made it legal to manufacture and sell cryptographic equipment and software domestically, but required a special license from the State Department to export any product that incorporated the technology to any foreign destination. Even making free encryption software available for download over the Internet required obtaining an arms control license.

Make strong encryption that we can't break, the government made clear, and you will never get a license to export your product to the world market. Anyone violating these rules could be prosecuted under the same laws that apply to arms traffickers and smugglers. That stick was more than large enough to discourage the industry from incorporating strong encryption into their products and standards, even as the need for it was increasingly recognized.

But in April, 1993, the government dangled a carrot next to the arms control stick: strong encryption that could be incorporated into products and that could still be freely exported. The system, called *key escrow*, aimed to provide a

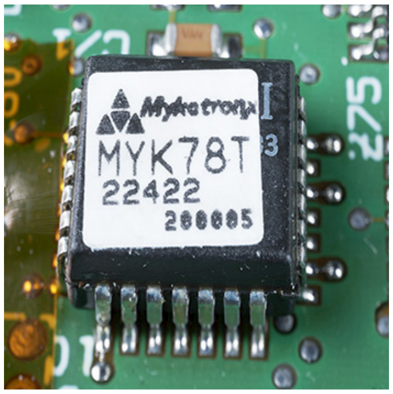


Figure 1: Mykotronx MYK-78T "Clipper" Escrowed Encryption Chip (photo courtesy of the author)

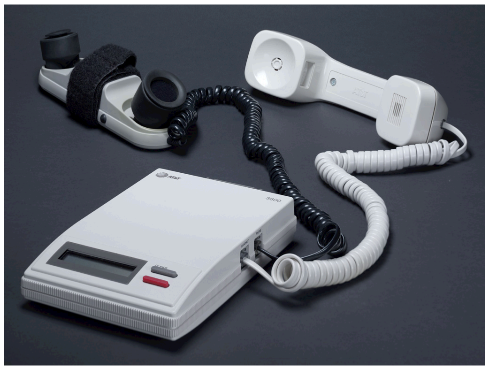


Figure 2: AT&T TSD-3600 Telephone Security Device (photo courtesy of the author)

strong cipher for public use with a "back door" that could be exploited by law enforcement agents conducting a (presumably authorized) wiretap.

The centerpiece of the key escrow proposal was a tamper-resistant hardware encryption module, called, in its initial version, the *Clipper Chip*. Clipper was intended as a drop-in replacement for a standard DES chip, but with a new symmetric-key cipher algorithm, called *Skipjack*, designed by the National Security Agency and using an 80 bit key. But before any two Clipper chips could communicate, they would first exchange a *Law Enforcement Access Field (LEAF)* that contained a copy of the current session key, itself encrypted with an "escrowed" *Unit Key* held by the government. Any Clipper-encrypted communication could thus be decrypted by government agents without needing to break the (presumably strong) Skipjack cipher. The agents would be able to recover the session key simply by decrypting the copy in the LEAF (which they would intercept along with the ciphertext) using their escrowed copy of the unit key. The system would, however, still be secure against *unauthorized* eavesdroppers, who presumably would lack access to the government's escrowed key database. Clipper chips (and other escrowed encryption modules in the same family) were to be manufactured, under a government license, by Mykotronx and available for sale to vendors of computers and other secure communications hardware; see Figure 1.

It was, the policymakers must have thought, a perfect solution.

2. BLOWING THE LEAF

Key escrow was not greeted by the technical community with the unreserved warm reception for which the government was perhaps hoping. Almost immediately, many objections were raised that questioned basic assumptions behind the proposal. Why should the bad guys be expected to use an encryption system that the government has announced in advance it can decrypt? How will the key escrow database be secured against unauthorized access? Why should industry adopt expensive hardware encryption (as Clipper required) just as software cryptography was becoming computationally feasible? Why should anyone trust the Skipjack cipher, an unpublished algorithm designed in secret by the NSA? And would the system reliably even solve the problem it aimed to address – ensuring government access to Clipper-encrypted traffic?

The history of the 1990's "crypto wars" has been well-chronicled, and it is beyond the scope of this short paper to address all the various problems, theoretical, practical, and political, with key escrow as it was envisioned and as it evolved. Instead, I will offer here a personal perspective, focusing on one small battle at the beginning of this (bloodless but still damaging) "war." I am surely omitting many important episodes, details, and contributors, for which I apologize; what follows should be understood as a war story, which is at best an idiosyncratic, personal, recollection.

2.1 The AT&T Connection

AT&T (my employer at the time) was the first (and ultimately the only) company to produce a full product based on the ill-fated escrow system, but that was not their original plan. The AT&T TSD-3600D, announced in 1992, was to be a voice encryption device that could be installed in any standard wireline "POTS" telephone (between the phone base and the handset). Calls placed to other TSD-3600D-equipped telephones would be automatically digitized (at 4800bps) and encrypted using DES, making eavesdropping on the conversation (by legal or illegal means) effectively infeasible under the technology of the time. The devices weren't cheap, but were designed by the same AT&T business unit that produced the STU-III secure telephone for the government, from which it borrowed some of its design and security features. Two communicating TSDs would first perform a Diffie-Hellman key exchange (768 bit, in the first version of the product) to establish a session key, a 4 character hash of which was displayed on each unit's LCD. To detect "man-in-the-middle" attacks, users could verify (by voice over the encrypted session) that their displayed hashes matched. See Figure 2.

When the US government learned of AT&T's plans to market the TSD, it worried that criminals might use the devices to thwart wiretaps. Plans for a new escrowed encryption system – with a wiretap backdoor – were hurriedly drawn up by the NSA, and AT&T was persuaded to replace the regular (non-escrowed) DES-based encryption scheme in the original TSD product with one based on the new system, which became known as the Clipper chip. In 1993, when Clipper was announced, a new Clipper-based TSD, dubbed the TSD-3600E, was announced at the same time. As incentive for AT&T's cooperation, the government agreed to purchase a significant quantity of Clipper-equipped TSD-3600Es, which sold for over \$1000 each. The original unescrowed DES-based TSD-3600D models were recalled by AT&T and quickly disappeared from the market.

When key escrow – and AT&T's involvement in it – was made public, I was just starting my career as a cryptography and security researcher in AT&T's Bell Laboratories division. My colleagues and I, like most members of the civilian cryptography research community, learned about the escrow scheme from the *New York Times,* and we were as skeptical as anyone of the security and practicality of the government's plan. Working for a company that was so prominently involved in what seemed like such a technically ill-advised project was a bit uncomfortable, but it also had its advantages. It was easier to get questions answered, to sort out how this technology was supposed to work. There might even be an opportunity to do some interesting research on the subject. After some poking around, I managed to get hold of a pair of first generation TSD-3600s, but this was less useful than I had hoped, especially given how infrequently I needed to have sensitive telephone conversations. The real breakthrough came when a group from NSA visited the Labs to brief us and answer our questions, which was especially helpful given the dearth of solid publicly released information on the technology. My colleague Steve Bellovin and I both took notes.

As the NSA meeting was breaking up, we asked, half-jokingly, if they'd mind if we posted a summary of what they'd told us to Usenet. To my great surprise, they enthusiastically agreed (evidently they were as eager to get details out as we were to learn them). Steve and I compared notes, and a few days later we posted a short writeup to the `sci.crypt` newsgroup.

In writing the summary, we were careful to stick to the facts, avoiding needlessly inflammatory commentary on the wisdom of key escrow or on whether the NSA should be trusted. This must have come as something of a relief to the NSA's readers of `sci.crypt`, flooded as it was at the time with relentless criticism of the Clipper program and of the government's intentions. A week later, the NSA invited me to come down to their headquarters and R&D facility at Ft. Meade, MD.

To make a long story short, I ended up bringing home samples of a next generation key escrow device. This was a PCMCIA card, code-named *Tessera*[1], intended for secure PC email and other data encryption applications. The Tessera card was based on a version of the key escrow chip, called Capstone, that added some public key exchange and digital signature features to its interface but was otherwise similar to the Clipper chip in its functionality. The NSA people asked only that I play with it and perhaps find interesting applications. I asked if I'd be able to publish my results, and, again to my surprise, they agreed.

2.2 Oracles and Rogues

As a research platform, the Tessera PCMCIA card offered a significant advantage over the Clipper-based TSD-3600 product: an open API with direct access to the encryption chip's functional interface. I could simply connect the card to a reader on my computer and write software to send data directly to and interrogate results directly from the Capstone chip (which included all the functions of the Clipper chip). This would be a much easier way to experiment with key escrow than the other alternative available to me, which involved removing the Clipper chip from a TSD-3600, reverse-engineering its pinout, and building and debugging an interface to it for my computer. With the PCMCIA card, all that was already done.

So I could get right to work. Which, of course, raised the question, to work on what, exactly? What questions would be interesting to ask about key escrow?

At the time, most of the questions and criticisms of the government's key escrow proposal were either political ("why should we trust the government to hold our keys?") or required access to classified information ("how does the Skipjack cipher work?"). But I decided to start with a question that the hardware might be helpful in answering: how do key escrow devices enforce the escrow features? That is, how do they prevent someone from using the Skipjack cipher without sending the data fields that enable a government eavesdropper to recover the key?

I found that the system included a number of safeguards to frustrate some of the most obvious ways to defeat the key escrow field, called the LEAF ("Law Enforcement Access Field").

The details of the LEAF were classified. We knew it was a 128 bit structure containing enough information for law enforcement recovery of the session key with the cooperation of the agencies holding the escrowed unit key database. We were told that the LEAF package contained a 32 bit unique unit identifier (the serial number of the chip that generated the LEAF), the current 80 bit session key (encrypted with the device's unit key) and a 16 bit LEAF checksum. The entire structure was encrypted with a "family key" to produce the final LEAF package. All cryptographic operations in LEAF creation were based on symmetric (secret) key techniques using the Skipjack cipher and other (unspecified) algorithms. The family key was global, shared by all interoperable key escrow devices (but not published). The Skipjack algorithm, the family key, the encryption modes used to encrypt the unit key and the LEAF message, and the details of the checksum algorithm were not made public (and all were protected against reverse engineering by Clipper's tamper-resistant hardware)[2]. Externally, the LEAF was presented as an opaque 128 bit package.

To decrypt escrowed traffic, a law enforcement agency first must intercept the LEAF and the traffic itself using some sort of data wiretapping technology. The LEAF could then be decrypted with the family key, revealing the chip serial number, the unit key-encrypted session key and the LEAF checksum. The target's serial number would then be provided by the agents to two "key escrow agencies," which would each return a "share" of the escrowed unit key associated with the given serial number. The two unit key shares would then be combined (by bitwise exclusive-or) to produce the full unit key, which the law enforcement agency could then use to decrypt the session key. This session key could in turn decrypt the actual intercepted traffic.

[1] *Tessera* was an unfortunate name. It turned out to be a registered trademark of a company that made PCMCIA cards and that wanted nothing to do with key escrow. The NSA eventually had to change the code name to *Fortezza.*

[2] In 1998, after Clipper was abandoned, the NSA declassified and published the Skipjack algorithm. I believe it was, and remains, the only NSA-designed symmetric-key encryption algorithm ever publicly released.

The key escrow system thus relied on the availability of the LEAF along with the encrypted traffic. To force applications to send the LEAF on the same channel as the traffic, key escrow devices would not decrypt data until they had received a valid LEAF for the current session key. Presumably, the chips on each end would perform various integrity checks on received LEAFs prior to accepting them.

To provide a convenient application interface for LEAF management, the devices generated and loaded LEAFs as part of the process of generating and loading the initialization vectors (IVs) for each cryptographic session. The Clipper and Capstone chips provided *generateIV* and *loadIV* functions that operated on 192 bit parameters instead of the usual 64 bits. This "IV" parameter was actually a two part structure containing the standard 64 bit IV concatenated with the 128 bit encrypted LEAF. The *loadIV* operation would fail if the LEAF did not pass an integrity check.

Most details of the LEAF creation method, encryption modes, and data structures, beyond those mentioned above, were classified and were therefore unknown to me. In particular, the key escrow standard did not specify the mechanism that enforced the transmission of the correct LEAF as part of the ciphertext stream. However, I was able to perform a number of simple experiments on my Tessera card to confirm and expand what we knew about the LEAF's internal structure and the way it was used. I found:

- LEAF integrity was verified via redundancy in its internal checksum field. In general, attempts to load an incorrect LEAF failed. This must have been due entirely to the checksum field and not through direct verification of the unit ID or encrypted session key fieds; the receiving chip could not confirm the correctness of those fields since it would have no way of knowing the unit ID or unit key of its peer. Therefore, the LEAF must have been testable based entirely on session information known to the receiver (such as the cleartext session key and IV) and that must have been included in the LEAF checksum computation.

- LEAF checksum computation included (implicitly or explicitly) the current IV. The LEAF changed whenever a new IV was generated even when the session key remained the same. Since the IV was not included directly as one of the LEAF fields, the only field it could affect would be the LEAF checksum. Furthermore, receiving devices would refuse to load a LEAF with the wrong IV.

- LEAF checksum computation must have included the cleartext of the current session key. Attempting to load an otherwise valid LEAF (and corresponding IV) from a previous session key failed. It was therefore not possible to "re-use" a LEAF generated from an old session key, even though such a LEAF would itself be internally consistent.

- The LEAF integrity check included every bit of the LEAF. Attempts to load an otherwise valid LEAF with a single bit inverted anywhere in the 128 bit structure always failed.

- The LEAF encryption method diffused its input across the entire 128 bit structure. The LEAF structure or encryption mode was apparently not exactly as specified in publicly released documents. Generating a new IV for a given session key caused changes across the entire LEAF. Since the Skipjack cipherblock size was 64 bits, encryption of the LEAF would have to involve at least two block encryption operations. Since the IV affected only the checksum, and the checksum appeared at the end of the LEAF structure in public documents, we could conclude that at least one of the following was true:
  - The LEAF was encrypted with a non-standard mode in which cleartext in "late" blocks affects the early ciphertext.
  - The LEAF was encrypted with a standard forward-chaining or stream mode but the checksum appears in the first cipherblock of the LEAF.
  - The LEAF was encrypted with a standard forward-chaining or stream mode but the current session IV was itself used to initialize it.

- The LEAF checksum was, in fact, 16 bits long. A brute-force search of the LEAF space for a valid LEAF required about 2^{16} operations.

That last point turned out to be interesting. It meant that it was possible to use 2^{16} Clipper or Capstone chip operations as an "oracle" to generate apparently valid, acceptable LEAFs for the current IV and session key that would actually be useless for escrowed decryption.

So the safeguards that required transmission of a valid LEAF weren't very strong after all. With only access to the chip's standard interface, one could easily create a "rogue" device that could happily interoperate with legitimate escrowed peers, enjoy the use of the strong Skipjack cipher, but be impervious to the key escrow back door. The only thing stopping you was a 16 bit exhaustive search, a very low barrier even in 1993.

In April, 1994, I wrote a paper about all this, "Protocol Failure in the Escrowed Encryption System". I circulated it to a few colleagues, and, not wanting to blindside anyone, also sent a copy to my contacts at NSA. They were, I must say, extremely good natured about it.

Eventually I submitted the paper to the upcoming 1994 ACM *Computer and Communications Security Conference*, which would be held in November. But some time in May, someone (I never found out who) sent a copy to John Markoff, the technology reporter at the *New York Times* who had broken the key escrow story the previous year. He called me to tell me he was writing a story about my paper for his paper, and wondered if I had any comment.

2.3 No Such Thing As Bad PR?

After the *Times* called, it occurred to me that I was in what could be considered an uncomfortable position, an employee of the research division of the same company in whose product I was finding flaws. And it was all based on a controversial wiretapping system created by a secretive government intelligence agency. And now the *New York Times* was about to write about it. And there I was, right at the center of the story. It seemed like a good time to involve management.

I feared that the company might not be completely delighted with my discoveries, or with my writing a paper on the subject. And indeed, executives in parts of AT&T couldn't understand why some kid in the troublemaking, out-of-control research lab would even think that it was a good idea to publish such things. But the Bell Labs management shined. They actively defended the importance of publishing and fully supported me as a member of the public research community, no matter the effect it might have on sales of the TSD or the company's relationship with the government. Our job as scientists, they argued, was to tell the truth. I was never prouder to work there.

Eventually, Markoff called me to let me know that his story would be running in the *Times* the next day. But when I got my copy of the paper, I couldn't find any mention of it. It was only later that I noticed the story in the one place I didn't look: the top of the front page, under the headline *Flaw Found in Federal Plan for Wiretapping.* Apparently cryptography was bigger news than I thought. More likely, it was a slow news day.

3. POSTSCRIPT

Clipper and key escrow eventually faded away. While my paper may have helped accelerate its demise, key escrow would not have been likely to succeed even if the Clipper escrow mechanism had been more robust than it was.

Fundamental problems with the government's vision for key escrow made it inherently unworkable, regardless of the implementation details. First, of course, was the problem of securing a growing database of end-user keys, a very attractive target for unauthorized eavesdroppers who might seek to intercept escrowed traffic themselves. Then there was the economic problem: communications cryptography was, by the 1990's, becoming an essentially zero-marginal-cost technology, something that could often be implemented in software more easily than by adding specialized hardware. But Clipper required the use of hardware cryptography, taking something that was becoming inherently cheap and turning it back into something expensive. The market would ultimately never accept this, even if the trust issues and technical problems could have been worked out.

Over the next few years there were attempts to revive key escrow under various new proposed schemes (the name eventually changed to "key recovery"). By the end of the decade, however, the government gave up. The export rules – the government's main leverage in promoting key escrow – were relaxed to allow mass-market products to use strong cryptography without a special license, and eventually, cryptography started to become integrated into more and more products, software, and protocols. Key escrow was finally dead.

It's probably worth asking whether this was a good thing. Law enforcement, after all, warned that unfettered access to cryptography would make it easier for criminals, spies, and terrorists to cover their tracks.

Fortunately, the worst fears of law enforcement haven't come to pass. Every year since the Nixon administration, the federal government has issued a report on legal wiretaps, giving the number of intercepts, the types of crimes being investigated and other statistics. Since 2002, the report has included another statistic: the number of cases in which encryption encountered on a legal wiretap prevented law enforcement from getting the evidence it was seeking.

With the increasing proliferation of eavesdrop-thwarting encryption built in to our infrastructure since export laws were relaxed a decade ago, we might expect law enforcement wiretap rooms to have become quiet, lonely places.

But maybe not. The latest wiretap report identifies a total of just six (out of 3194) cases last year in which encryption was encountered, and this prevented recovery of evidence a grand total of *zero* times.

What's going on here? Shouldn't all this encryption be affecting government eavesdroppers at least a little bit more than the wiretap report suggests? Do the police know something about cryptanalysis that the rest of us don't, enabling them to effortlessly decrypt criminal messages in real time without batting an eye? Is AES (the federally-approved algorithm that won an open international competition for a new standard block cipher in 2001) part of an elaborate conspiracy to lull us into a sense of complacency while enabling the government to secretly spy on us? Perhaps, but the likely truth is far less exciting, and ultimately, probably more comforting.

The answer is that faced with encryption, capable investigators in federal and local law enforcement have done what they have always done when new technology comes around: they've adapted their methods in order to get their work done. Widespread encryption, rather than shutting down police wiretaps, has actually pushed them in a more reliable – and accountable – direction.

This is because while traffic encryption is highly effective at preventing wholesale, *un-targeted* interception, it does remarkably little to prevent *targeted* government eavesdropping in the complex architectures of modern computing and communication technologies. Yes, today's encryption algorithms are believed to be effectively secure in practice, in the sense that they make it infeasible for even an adversary with the resources of a government to obtain cleartext from ciphertext without access to the key. But government eavesdroppers doesn't have to limit themselves to that scenario for their wiretap targets. They can instead exploit the reality that the cleartext (or the keys to decrypt it) for almost all encrypted traffic today is typically available, somewhere, on a general-purpose computer that is exposed to government access, either explicitly or through surreptitious means. And as systems become more sophisticated and incorporate more features, the exposure of cleartext and keys to third party access tends to increase correspondingly. All without Clipper chips or any other kind of key escrow systems, mandated or not.

If only we had understood that in 1993. We could have saved ourselves a quite a bit of trouble, and maybe spent a bit more time actually making things more secure.

Reliable Telemetry in White Spaces using Remote Attestation

Omid Fatemieh
University of Illinois
Urbana-Champaign

Michael LeMay
University of Illinois
Urbana-Champaign

Carl A. Gunter
University of Illinois
Urbana-Champaign

ABSTRACT

We consider reliable telemetry in white spaces in the form of protecting the integrity of distributed spectrum measurements against coordinated misreporting attacks. Our focus is on the case where a subset of the sensors can be remotely attested. We propose a practical framework for using statistical sequential estimation coupled with machine learning classifiers to deter attacks and achieve quantifiably precise outcome. We provide an application-oriented case study in the context of spectrum measurements in the white spaces. The study includes a cost analysis for remote attestation, as well as an evaluation using real transmitter and terrain data from the FCC and NASA for Southwest Pennsylvania. The results show that with as low as 15% penetration of attestation-capable nodes, more than 94% of the attempts from omniscient attackers can be thwarted.

1. INTRODUCTION

Dynamic spectrum allocation promises to make spectrum use more efficient by enabling opportunistic, unlicensed (secondary) use of 'white-space' frequencies when they are not occupied by licensed (primary) users. This paradigm has gained significant traction due to the increasing demand for wireless services, the limited availability of spectrum, and the FCC's recent ruling that permits operation of unlicensed users in the unused portions of the TV spectrum. This permission is considered the first significant increase in unlicensed spectrum below 5 GHz in over 20 years [2].

Identifying unused portions of spectrum is a key requirement for opportunistic spectrum access. Spectrum availability data is envisioned to be centrally aggregated and consulted to govern the usage of spectrum. At least three scenarios for data collection have been proposed. First, the data may be provided by volunteer white-space devices or deployed sensors to build regional or nationwide spectrum availability databases, or augment the white space geolocation database mandated by the FCC. Second, a white-space service provider may collect spectrum sensing data from white-space devices in its network to determine areas of primary presence [1, 4, 11]. Third, by combining spectrum sensing data from multiple devices (*collaborative sensing*), one can improve the detection accuracy in highly shadowed environments [37].

Reliable Telemetry, or reliable central aggregation of sensor data in this context, is threatened by nodes that may report false data with malicious intent. A coordinated group of attacker nodes may aim to *exploit* a spectrum in a given region by falsely reporting that a primary signal is present, or *vandalize* a primary by reporting that its signal is not present and thereby creating interference and chaos. Previous works on this problem have achieved moderate degrees of success by identifying the data from individual or small groups of attackers as abnormal. These approaches suffer from at least two shortcomings. First, their effectiveness is limited by relying solely on sensor data for inferring (dis)trust. Second, they assume limits on the penetration of attackers in an area; attackers either constitute a small fraction of nodes in a small local neighborhood [30], or if they control the majority of nodes in a neighborhood, the preponderance of adjacent neighborhoods must be un-compromised [18].

In this paper, we initiate a new direction in reliable distributed measurement by relying on a small subset of nodes that can perform *remote attestation*. These nodes can securely attest their operating state to a remote server. They will be excluded if they are detected as compromised. Otherwise, they will be used as a foundation for security and reliability. To that end, we propose a practical framework for using data from both attested and regular nodes to deter attacks, while achieving quantifiably accurate results in the absence of attacks. More specifically, we explore a strategy based on statistical sequential sampling and inference to obtain an estimate for signal power in each small region. The sampling method uses data from all of the attested nodes, as well as the minimum required data from the rest of the nodes to achieve accuracy with a pre-specified margin of error. Next, the data contributed by non-attested nodes is verified against data from attested nodes in the neighboring areas. This step is performed using SVM classifiers with quadratic kernels that are trained with an initial set of trusted signal propagation data in the region of interest.

We evaluate our scheme using predicted signal power data obtained from applying empirical signal propagation data to real-world TV transmitter and terrain data from the FCC and NASA databases. We instantiate the evaluations to a hilly urban/suburban area in Pennsylvania and measure the performance of our approach in the absence and presence of omniscient coordinated attackers. In addition, we systematically enumerate the costs associated with remote attestation and provide detailed data on these costs for prototypes based on Trusted Platform Modules (TPMs) and AVR32 microcontrollers. The data shows attestation may introduce non-trivial costs, which motivates our approach to leveraging attestation efficiently to establish trust in spectrum sensing results. Our evaluation results show that our scheme is highly effective against attacks even in cases where only a small subset of the sensors can be remotely attested. For example, with as low as 15% of nodes being attested, we show that more than 94% of the attacks can be defended against. The protection gradually improves as the fraction of attested nodes increases.

The contributions of this paper are summarized below:

- A new direction in reliable telemetry against coordinated misreporting attacks that relies on a small subset of attestation-capable sensors.
- A practical framework for using statistical sequential estimation coupled with machine learning classifiers to deter attacks and achieve quantifiably accurate outcome.
- A case study based on real TV transmitter and terrain data from the FCC and NASA in Pennsylvania that includes an evaluation for the proposed scheme, as well as a cost analysis for remote attestation.

2. BACKGROUND AND PROBLEM FORMULATION

In this section we first provide background information on spectrum measurements and remote attestation. Next, we describe our setting and problem statement.

2.1 Spectrum Sensing and Aggregation in White Spaces

The FCC's ruling in November 2008 allows for operation of unlicensed users in the unused portions of the TV spectrum [2]. Wireless communications in this spectrum (below 700 MHz) benefit from favorable signal propagation and penetration properties, which enable long transmission ranges. Access to this spectrum could enable more powerful Internet connections in public areas, campuses, and homes with extended range, fewer dead spots, and improved speeds. Many other applications are envisioned; for example, broadband access for rural areas, extended access to medical care in rural areas, and support for the communications of the advanced meter infrastructure (AMI) [3, 17].

Sensing the spectrum to identify unused channels can be used to improve the performance of white space networks. This is despite the FCC's September 2010 ruling which exempts the devices that incorporate geo-location and can access a new TV band database from mandatory spectrum sensing [3]: (1) The ruling still allows for operation of sensing-only devices that cannot or do not access the database. (2) The database is built from conservative propagation models, which results in declaring many unused channels as occupied in places far from the transmitters. Real-time spectrum sensing data can provide a more accurate view of spectrum availability, or be used to improve the database results. (3) In places where multiple channels are available, the spectrum sensing details can reveal the highest quality channels for communications.

There exist three scenarios for centrally aggregating spectrum sensing results from sensors in a large region [19]. First, using data from deployed spectrum sensors or volunteer (mobile) white-space devices to build a regional or nationwide spectrum availability database. Such a database can be used to augment the white space geo-location database mandated by the FCC, or to learn spectrum usage as part of the recently passed Spectrum Inventory Bill [6]. Second, a white-space service provider or base station may collect spectrum sensing data to determine areas of primary presence from cognitive radios in its network. This centralized approach has been endorsed by the IEEE 802.22 WRAN standard draft [4], CogNeA [1] and recent research prototypes [11]. The spectrum sensing data collected by the service provider may be provided by not only in-network cognitive radios, but also deployed spectrum sensors, and additional volunteer (mobile) devices to determine areas of primary presence. Third, spectrum sensing results from multiple devices may be combined to improve the detection accuracy at low thresholds in highly shadowed environments [37].

To capture the common aspects of the above scenarios, we focus on the case of building a regional spectrum availability database by a service provider. The database may then be combined with databases from other regions to form a nationwide database of spectrum sensing. The spectrum sensing data used to populate the database is provided by one or more of the following sources.

- *Volunteer Radios:* a set of (mobile) devices with different owners. The data would be collected by a modern 'mobile app' built to perform spectrum sensing at its current location and report the results to a central server. This form of participatory sensing is also referred to as *crowdsourcing*.
- *In-Network Cognitive Radios:* cognitive radios that are part of the service provider's network.
- *Dedicated Sensors:* sensors (in the form of a wireless sensor network) deployed for the specific task of spectrum sensing alongside the main white-space network [16].

2.2 Remote Attestation

Remote attestation is a technique for a system to provide certified information about its operating state (*i.e.* software, firmware, or configuration) to a remote party. This process is typically initiated by a request from the remote party. Upon receipt of the request, the queried system creates a (signed) record of the system's operating state and sends it to the initiator. To securely record and certify its current state, the system needs to contain a number of components. Trusted hardware components are often used to this end, although software can also be used in some cases. Regardless, remote attestation imposes additional computational, storage, energy, time, and potentially manufacturing costs on both parties. On desktop PCs, the Trusted Platform Module (TPM) is often used to provide remote attestation functionality. The Trusted Computing Group (TCG) is developing trusted computing standards specifically for mobile devices to minimize costs and support appropriate usage models, and have specified several primitives for a Mobile Trusted Module (MTM). MTMs are expected to be available for many new mobile applications in the near future [7]. Previous work has also shown that remote attestation can feasibly be implemented in software on-chip for embedded processors such as AVR32 micro-controllers [25].

2.3 Setting and Problem Statement

We consider building a spectrum availability database from received signal power data from a combination of volunteer radios, in-network cognitive radios, and deployed sensors. We refer to these sources as nodes or sensors in the rest of this paper. Due to their widespread adoption, ease of implementation, and small sensing time, we assume that energy detectors will be the only sensors in use [8, 36]. We also assume the primary signal faces path loss and shadow fading due to irregular terrain and obstacles such as trees, buildings, walls, and windows.

The spectrum availability database represents the region of interest as a grid of small cells (or *tiles*) on the map of the region. Each cell may be a 1km $\times$ 1km square and is the unit in which combining individual results, or *collaborative sensing*, occurs. Within a cell, we combine the raw signal power measurements from nodes to determine primary presence (as opposed to binary yes/no results).

This allows for using signal power as a measure of quality among the available channels and enables us to detect misreporting attacks. A common method for combining sensing results within each cell is Equal Gain Combining (EGC), which periodically averages the power measurements of individual nodes in each frequency channel and compares it to a *detection threshold* λ. In the case of primary Digital TV (DTV) transmitters, FCC has mandated -114 dBm as the detection threshold.

We address the problem of performing reliable aggregation of spectrum measurement data contributed by a distributed set of nodes. An attacker may compromise a (large) subset of the nodes and make them act in cooperation in order to change the spectrum sensing outcome in any cell, including any number of adjacent cells. For example, they may seek to change the perceived primary signal power for a cell from a value below threshold (*-120 dBm*) to a value above threshold (*-100 dBm*), or vice versa. The first attack is called *exploitation*, and the second is called *vandalism*. In exploitation, the attackers aim to deceive the network to abandon the channel to exclusively use it for themselves, whereas in vandalism the main goal is creating chaos or interference. We focus on canceling the effect of such attackers that have a strong (*e.g. majority*) presence in a cell, and (in the absence of any defense) are able to dominate the cell and flip the detection outcome.

A particularly novel aspect of our work is that we assume that a subset of nodes, for example 20%, are able to perform remote attestation (see Figure 1). For any such *attestation-capable* node, the aggregation server can detect whether it is compromised and thus running illegitimate code. The question that we aim to answer is how to efficiently and effectively use this capability to obtain reliable spectrum sensing results. This question is particularly important when the attestation-capable nodes constitute a small fraction of the nodes. This may be due to the low penetration of the technology among the volunteer nodes, or cost considerations of deploying *and* using this capability by the service providers in the deployed sensor scenarios (see Section 5).

While some of the nodes may be unreliable or compromised insiders, we assume that each node maintains a secure link to the base station for sending spectrum sensing results, and that attackers are unable to fabricate nodes or identities arbitrarily ('Sybil' attacks [32]). The secure links can be realized using pre-shared keys or a PKI, which may also serve as a foundation for preventing Sybil attacks by being associated with the identity of each node. Alternatively, one can take the dual view that we aim to demonstrate a method that forces adversaries to discover and deploy a practical Sybil attack, which requires a higher level of sophistication on the attacker's side (*e.g.* faking multiple link layer addresses). We also assume that the locations of nodes are reliably known through GPS or other localization techniques and nodes do not misreport their locations. This assumption is easily achievable in two of the most popular proposed applications of white space networking that assume fixed nodes with known locations: (1) Residential Internet access using IEEE 802.22 [36], and (2) AMI communications [17]. In cases where the network contains untrustworthy or mobile devices, secure localization and location verification techniques may ensure nodes' locations are authentic [14, 24, 26, 27]. The above assumptions are common for the type of analysis we perform here [15, 19, 31]; if they are violated then additional protective measures are required.

3. APPROACH

Consider Figure 1 as part of the region of interest for performing reliable aggregation of spectrum measurement data. There exist two types of nodes; *attestation-capable* nodes (triangles), and *regular* nodes (circles). In any particular cell, the goal is to obtain an estimate of the signal power in that cell, and compare it to a primary detection threshold to determine whether the spectrum is unused. Assume for now that we have performed remote attestation on all attestation-capable nodes and have excluded those we believe are compromised. Therefore, the remaining attestation-capable nodes are considered *trusted* or *attested*. For regular nodes, however, we do not have any prior information regarding their legitimacy.

Consider cell A in Figure 1 in which about half of the nodes are attested. One may argue that the high number of reliable nodes provides enough diversity to absorb the variations due to path loss and shadow-fading, and therefore there is no need to include the results of regular nodes. This approach is safer (in terms of vulnerability to false reports) than one in which the values from the (potentially compromised) regular nodes are also included. But what if the rest of the regular nodes are also legitimate? Is the safety worth the reduced precision? How would we determine whether it make sense to rely only on trusted nodes, or we should use the data from regular nodes as well? And if so, which ones?

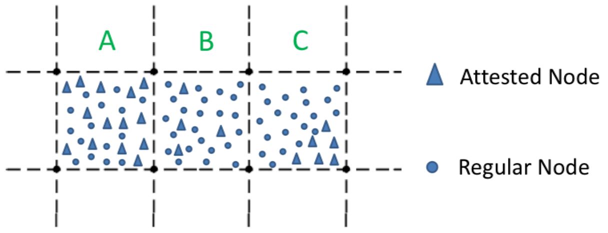


Figure 1: Illustration of a few cells with attested and regular nodes.

Now consider cell B where unlike cell A there are very few trusted nodes. Therefore, there is a high chance that aggregating the measurements from such a small number of nodes does not provide enough diversity to obtain a precise measurement (estimate) of the signal power. A similar situation can be seen in cell C; not only there exist very few attested nodes, but their positioning also makes it likely that they do not provide enough diversity. For example, they may all be behind an obstruction that attenuates the signal. Therefore, it seems necessary to include results from at least some of the regular nodes. But what if some or all of them are compromised, and they skew the results to achieve their malicious goal instead of adding legitimate diversity?

3.1 Key Issues and Overview

The examples above underline the importance of the following needs. First, there must be a systematic strategy to determine when there is enough diversity in the results that we can stop collecting additional data within a cell. Second, if we decide we need additional data beyond those from attested nodes, there should exist a strategy to decide which nodes to include. Third, for each cell in which additional regular nodes are added to the data 'pool,' we need a strategy to ensure that the added nodes are not dominated by attackers.

At a high level, our approach consists of three main phases (summarized in Algorithm 0). First, within each cell we rely on statistical inference and sequential estimation to aggregate data from all of attested sensors as well as 'enough' regular nodes to achieve the application-specified precision goal (Section 3.2). Note that we only include the least required number of regular sensors to limit unnecessary exposure to untrustworthy data. Various inclusion strategies are proposed for this purpose (Section 3.3). The aggregate is either the mean and median of the data, and is dynamically determined by our algorithm. This choice may change throughout the execution of the algorithm according to pre-specified rules (Section 3.4). Second, the regular nodes that were included in the aggregation process in the cell are compared against the data from the trusted nodes of the 8 neighboring cells. This process involves using machine learning classifiers built from real signal propagation data. The classifier detects irregular signal propagation patterns that most likely represent a coordinated misreporting attack (Section 3.5). Third, after the potentially compromised data is eliminated, we compute the final aggregate.

Algorithm 1 Simplified Approach Overview (for Each Cell)

```
Input:
(1) Green Data: measurements from attested nodes
(2) Yellow Data: measurements from regular nodes
(3) Strategy ∈ {Random, Geo-Diverse, Biased}: strategy for including data from regular nodes
(4) Aggregate ∈ {Mean, Median}: dynamically changes based on the situation

Phase 1: Node Selection
Add Green Data to aggregation Pool
while ¬SATISFY-PRECISION-REQUIREMENTS(data in Pool, Aggregate) do
    if | Yellow Data | > 0 then
        MOVE-NEXT-ELEMENT-TO-POOL(Strategy, Yellow Data)
    else
        Remove all Yellow Data from Pool
        Go to Phase 3
    end if
end while

Phase 2: Attack Detection
Yellow Suspects ← Yellow Data in Pool from Phase 1
Green Neighbors ← averages of Green Data in the neighboring cells (i.e. 8 numbers)
if SVM-ATTACKER-DETECTION(Yellow Suspects, Green Neighbors) then
    Remove all Yellow Suspects from Pool
end if

Phase 3: Aggregate Calculation
Compute Aggregate from data in Pool
```

3.2 Using Statistical Inference to Ensure Precision

For many applications, including aggregation of spectrum sensing data, it is not clear in advance how many sensors (observations) should be used in each aggregation effort in order to achieve the desired precision in the (estimation) outcome. Instead, data is evaluated as it is collected, and further sampling is stopped in accordance with a pre-defined *stopping rule*. This process is also referred to as *sequential estimation.* In our case, we aim to achieve an acceptable precision in the results while using as few data points from regular nodes as possible. We argue that sequential estimation for achieving fixed width confidence interval for the estimated aggregate is an ideal tool to achieve our goal. By stating the acceptable *margin of error* (half the width of a confidence interval) for the quantity being estimated, the application can ensure with high confidence that the estimated outcome from the sample data is 'close enough' to the true value. In other words, with high confidence (*e.g.* 95%), it can be assured that the true mean (or median) is within a γ margin of error from the estimated value (*e.g.* $\gamma = 3dB$). This is also referred to in the form of a *coverage probability* (*e.g.* $0.95 = 1 - \alpha$).

We first focus on a sequential procedure for finding fixed-width confidence intervals for the mean. Let $x_1, x_2, ...$ be a sequence of independent and identically distributed (i.i.d.) random variables having an unknown density function $f(x), x \in R$. The i.i.d. assumption is not absolutely true for sensors that are very close and face correlated shadowing; however in view of practical considerations we proceed with this assumption, which is in-line with the commonly used log-normal shadowing model [33]. Let μ and σ^2 represent the mean and variance of density function $f(x)$. It is known that no fixed-sample size procedure will provide a fixed-width confidence interval for μ having a prescribed coverage probability at the same time. The famous Chow-Robbins procedure for sequential estimation defines the following stopping rule for a confidence interval of size 2γ:

$$N = \inf\{n \geq n_0, n \geq a^2\gamma^{-2}s_n^2\}$$

where $n_0 \geq 2$ is the initial sample size, $a = z_{(1-\alpha/2)}$ is the $100(1-\alpha/2)$ percentile of the standard normal distribution $N(0,1)$ (*e.g.* if $\alpha = .05$ then $a = 1.96$), and s_n is the sample standard deviation of n observations. The Chow-Robbins procedure is asymptotically tight, in the sense that the coverage probability is asymptotically $1-\alpha$, and is also asymptotically efficient in the sense that the average required number of samples is asymptotically equal to an optimal fixed-sample procedure with *known* σ^2 [20].

Now we turn to the median. We begin by placing the measurements in order, that is: $x_{(1)} < x_{(2)} < ... < x_{(n)}$. The goal is to find an interval $x_{(a)} < m < x_{(b)}$ such that $P(x_{(a)} < m < x_{(b)}) = 1 - \alpha$, where $1 - \alpha$ is the desired probability that the interval captures the median.

In order to have $x_{(a)} < m$, at least a of the observations must fall less than m, and in order to have $m < x_{(b)}$, at most $b - 1$ of the observations must fall less than or equal to m. Since m is the median and since the distribution of the X's is continuous, we have

$$P(X < m) = P(X \leq m) = .5.$$

Assuming independent observations, the probability that at least a and at most $b-1$ of the observations fall less than m is given by the binomial probability with $p = .5$, that is $\sum_{k=a}^{b-1}\binom{n}{k}(.5)^n$. To construct a $100(1-\alpha)\%$ confidence interval for m, we choose a and b so that this sum is $1-\alpha$. For large samples, approximate values of a and b may be found by using the normal approximation to the binomial distribution. We may obtain a and b by solving for them in the following equations [22]:

$$\frac{a - .5n}{\sqrt{.25n}} = -z_{(1-\alpha/2)}, \quad \frac{b - 1 - .5n}{\sqrt{.25n}} = z_{(1-\alpha/2)}$$

Note that both the confidence intervals were calculated by assuming the distribution of the original population is unknown.

3.3 Intra-cell Inclusion Strategies

We consider three *inclusion strategies* for including regular nodes in the aggregate computation in each cell. The merits and disadvantages of each strategy are discussed in this section and evaluated in Section 4.

Random: Randomly adding data from regular nodes to the data from attested nodes has the advantage that it is in-line with the sampling assumptions made in computing the confidence intervals. In addition, the randomness reduces the attacker's chances of selectively compromising nodes and carefully crafting false measurements with minimum abnormality. However, it disallows deploying targeted inclusion strategies that could potentially lead to lower attacker success rate.

Geo-Diverse: By selecting a geographically diverse set of regular nodes, we add diversity to the results and reduce the chances of selecting (regular) nodes that are experiencing similar shadowing effects. To achieve this goal, we use the widely cited Gudmundson shadow correlation model [21]. According to this model, the correlation in shadow-fading in distance Δx is represented as:

$$R(\Delta x) = e^{\frac{-\Delta x}{d_{corr}}}$$

with the correlation length d_{corr} dependent on the environment. Empirical studies suggest values between $25m$ to $120m$ for urban areas [9]. Using this model, we suggest the following greedy approach to adding nodes to the aggregation pool. Before each addition to the pool, we compute the aggregate correlation of all nodes already in the aggregation pool with the candidates to be added to the pool. At each step, we add the node with the least aggregate correlation with existing nodes.

Biased: In this approach, we sort the data from the regular nodes in the increasing order of the absolute value of their difference to the median of the attested nodes. At each step, we move values to the aggregation pool according to their rank in the sorted list. This approach has the disadvantage that creates a 'bias' in the aggregate calculation process, which makes the computations in Section 3.2 inaccurate. However, in many cases, this bias effectively works as an implicit weighting mechanism in situations where attackers have only compromised a subset of the regular nodes. In those situations, this approach may limit the number of measurements from compromised nodes that will be included in the final result (see the results in Section 4).

3.4 Intra-cell Aggregation: Mean or Median?

Within each cell, the two main options for aggregating measurements in a cell are calculating the average (EGC) or median of the data (observations). A collection of observations is referred to as a sample. The goal is to use all of attested nodes plus a dynamically selected set of regular nodes such that we can ensure the computed aggregate is within a pre-defined distance of the real mean or median for the signal in the cell.

The median has a key advantage over the mean as an aggregate; it is less vulnerable to natural outliers or attacker nodes that constitute a minority of nodes in a cell [18, 38]. However, computing the sample median with a pre-specified confidence interval requires more data (compared to mean). Or dually, with a fixed number of observations, the confidence intervals achieved for the median are larger than those computed for the mean (the calculation procedures are presented in Section 3.2). To support our argument about the relatively smaller confidence intervals for mean (with the same number of samples), we generate sample signal propagation data representing a log-normal shadowing model with average power of -95dBm and standard deviation (*a.k.a.* dB-spread) of 4, 6, and 8. Table 1 presents the margins of error achieved using random samples of size 20, 30, 40, and 50 from this distribution.

However, if the attackers obtain even a weak majority in a cell, they can move the median to their desired number while being less 'abnormal.' Figure 2 illustrates this observation. The attackers' goal is to change the aggregate from a value below the signal threshold of -114 dBm to one above the threshold (*e.g.* -113 dBm). When the median is used (the top picture), the attackers can achieve their desired goal by simply reporting -113 dBm. However, when the average is used (the bottom picture), the attackers need to report an average false report of -105.5 dBm to change the total average to their desired value of -113 dBm. The additional abnormality

Table 1: Margin of error (95% confidence) for randomly generated data of size $|S|$ equal to 20, 30, 40, 50 from log-normal distribution with standard deviation, σ, of 4, 6, and 8.

	$\|S\| = 20$	$\|S\| = 30$	$\|S\| = 40$	$\|S\| = 50$
	$\sigma = 4$			
Mean	1.7	1.4	1.2	1.1
Median	2.3	1.9	1.6	1.5
	$\sigma = 6$			
Mean	2.6	2.1	1.8	1.6
Median	3.5	2.8	2.5	2.3
	$\sigma = 8$			
Mean	3.5	2.8	2.4	2.2
Median	4.7	3.8	3.3	2.6

facilitates detecting them using SVM classifiers (Phase 2 of our approach), and is therefore desirable. Hence, we will rely on median when the attested nodes represent the majority of nodes in the cell and rely on the mean otherwise. This strategy helps with reducing the effect of attackers and natural outliers when attackers do not constitute a majority, and makes them more likely to be detected when they do.

We further elaborate on the details of aggregate calculation in Phase 1 with an illustrative example. We start by considering the data from all of the attested nodes in the aggregation pool and initially use *median* as the aggregator. If the margin of error for the median of attested nodes is below the application requirement γ, we stop by declaring the median as the final result. Otherwise, we need more data. Consider a cell with k attested nodes. After adding the k attested nodes, we iteratively add up to $k - 1$ additional elements from regular nodes to the aggregation pool.

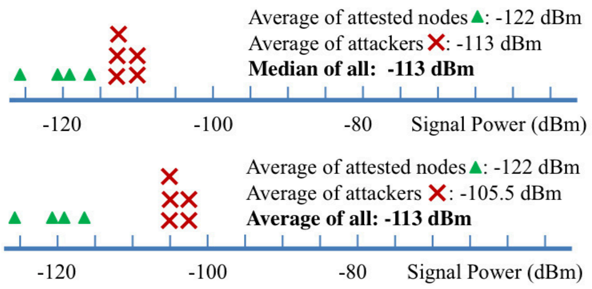

Average of attested nodes ▲: -122 dBm
Average of attackers ✕: -113 dBm
Median of all: -113 dBm
-120
-100
-80
Signal Power (dBm)
Average of attested nodes ▲: -122 dBm
Average of attackers ✕: -105.5 dBm
Average of all: -113 dBm
-120
-100
-80
Signal Power (dBm)

Figure 2: A simplified illustration of why attackers are forced to deviate more when they aim to move the mean (bottom picture) instead of the median (top picture).

The order in which the regular nodes are added to the pool is determined by the chosen inclusion strategy (Random, Geo-Diverse, or Biased). After each addition, if the margin of error for the median is reduced to a value lower than γ, we transition to Phase 2. If this condition is not met at any point and there exist additional measurements, we switch to using *mean* as the aggregator. Again, we continue adding new data from the regular nodes to the aggregation pool (using the same inclusion strategy) until the stopping rule is satisfied. If so, we transition to Phase 2. Otherwise, if adding all of the regular nodes does not result in satisfying the stopping rule, we ignore all the added regular nodes and proceed to Phase 3 where the median of attested nodes is computed as the aggregate.

3.5 Inter-Cell Attacker Detection using Classifiers

When the execution of Algorithm 0 reaches Phase 2, we have obtained an aggregate from data provided by all of the attested nodes, as well as *some or all* of the regular nodes in the cell. In this phase, we aim to ensure that the regular nodes whose data is included in the calculation are not part of an exploitation or vandalism attack. We first separate the data points from those *regular* nodes that have contributed to the aggregate (*a.k.a.* 'yellow suspects') and compare them to the data from *attested* nodes in the neighboring cells (*a.k.a.* 'green neighbors').

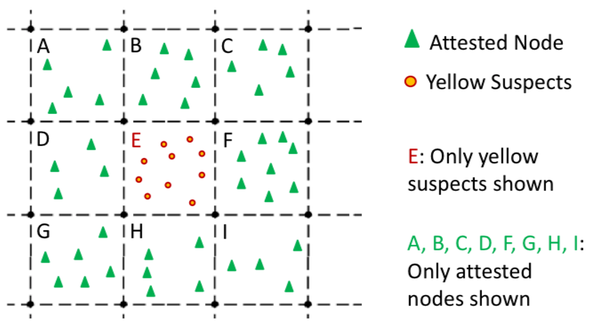

A
B
C
D
E
F
G
H
I
Attested Node
Yellow Suspects
E: Only yellow suspects shown
A, B, C, D, F, G, H, I: Only attested nodes shown

Figure 3: Classification-based attacker detection setting: regular nodes included in the aggregation for cell E and attested nodes from neighboring cells.

To determine if the yellow suspects in a cell represent an attacker-dominated group, we use real signal propagation data in the region to build a *classifier* that is *trained* to differentiate between natural and un-natural signal propagation patterns. The idea is to learn the normal propagation patterns of the signal from the reliable signal propagation data and use it to spot unnatural propagation of signal, which may be caused by malicious false reports.

More specifically, we consider the *local neighborhood* N_E of any cell E to contain E and its 8 neighboring cells (Figure 3). We represent N_E by a 9-element tuple containing the 'average' reported powers from the yellow suspects in E and the average reported power from the green neighbors in a pre-specified order. We call this the *neighborhood representation* of E. For example, for Figure 3, the first element would be the average of yellow suspects in cell E, and the second to ninth elements would be the value in the first element minus the average power of attested nodes in cells A to I (excluding E). Assume for a moment that we have access to reliable power measurements for a subset of the region of interest. This data can be used to create one neighborhood representation for each cell in the area. We refer to each such representation as an 'example.' Therefore, we can assume access to a large number of such examples representing the 'natural' propagation of signal in local neighborhoods. Also, as we will elaborate later, assume we have access to the neighborhood representation for a sufficiently large and diverse set of 'un-natural' (attacker-dominated) cells.

We now cast our problem to a binary *classification* problem. Classification is a machine learning technique that is widely used in domains ranging from spam email detection and unauthorized spectrum usage to fraud detection and speech recognition. In a binary classification problem we are given a set of *training* examples with their corresponding labels, $(\overrightarrow{x_i}, y_i)$, where $\overrightarrow{x_i}$ is the representation of the i^{th} example and $y_i \in \{1, -1\}$ ('yes' or 'no') is the corresponding binary label. Each example is described by a vector of its attributes which is often called the feature vector. In our case, the neighborhood representation of a cell serves as its feature vector. The goal is to predict a binary label for a *test* example for which we do not know the label, using the classifier built from training examples [12]. A classifier tries to partition the input feature space into regions where positive examples lie versus regions where negative examples lie. The boundary between regions for positive and negative examples is called the *decision boundary*. Training involves learning the decision boundary and classification involves determining on which side of the decision boundary a test example lies.

Now we turn to the problem of obtaining training examples. We argue that normal (negative) instances can be obtained in a practical one-time process based on a trusted sensor grid. By one-time we mean that in a particular region, we only need to collect signal propagation data once to build the classifier for that region. Once the classifier is built, it can be used forever (or until there is a significant environmental change in the region). A typical strategy for collecting this data is war-driving where a sensor is moved though the region collecting training data as it goes. Having obtained such natural (normal) examples, we modify them to inject *un-natural* training instances to represent attacker-dominated cells.

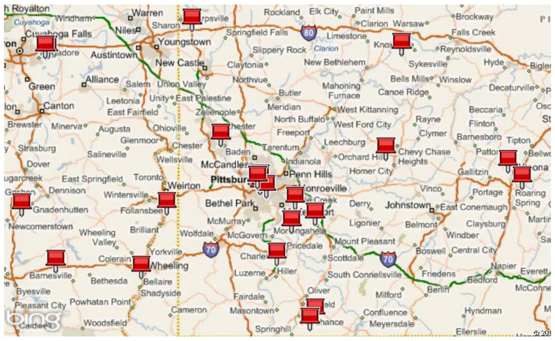
Building such a classifier from the natural and un-natural examples has been discussed in detail in prior works [19] and has been shown to effectively detect attacker-dominated regions in environments where there is no separation between regular and attestation-capable nodes. By contrast, in our setting, the classifier is applied in a slightly different manner where only the trusted data from the neighbors is used in classification. However, due to potentially low penetration of attested nodes, this translates to less data points being available for classification. This may negatively affect the classification accuracy. We build a similar classifier (using Support Vector Machines (SVM) with quadratic kernels) to detect whether the yellow suspects in a cell look abnormal compared to the green neighbors and evaluate it in Section 4. If the classifier considers the data to be anomalous, we only rely on the median of the attested nodes in that cell. Otherwise, the aggregate computed in Phase 1 (using a mix of attested and regular nodes) is valid and should be used as the representative signal power in that cell.

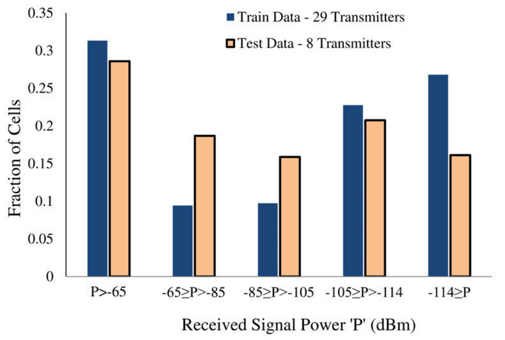
4. EVALUATION

We evaluate our system using predicted signal propagation data obtained from real transmitters and terrain data. More specifically, the TV transmitter location, signal power, height, and frequency is obtained from FCC databases and terrain (*i.e.* elevation for any given point) is obtained from NASA databases [5]. We choose the FCC-endorsed Longley-Rice empirical outdoor signal propagation model to generate predicted signal power for any location and frequency of interest. Longley-Rice takes into account the effects of terrain as well as transmitter's power, location, frequency, and height. To account for additional uncertainties due to factors such as shadow-fading we add log-normal variations with a mean of zero and a standard deviation (dB-spread) of $\sigma_{dB} = 6$ to the predicted signal power for each point [37]. For evaluation purposes, we consider this data as the ground truth.

We instantiated our evaluation to an urban/suburban area surrounding Pittsburgh, Pennsylvania. The hilly nature of the area introduces a large amount of legitimate signal variations, which makes the task of precise signal power estimation and attacker detection more challenging (compared to flat areas). Therefore, these experiments should be considered a stress-test for our scheme.

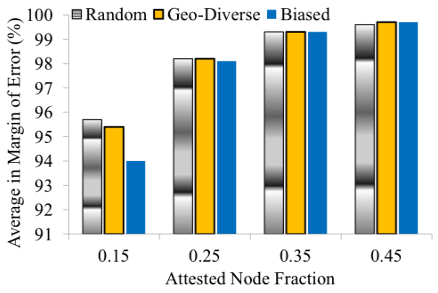
The following points in (*latitude, longitude*) format define the southwest and northeast corners of the considered 20km × 20km square area in Pennsylvania: ⟨ (40.35, -80.12), (40.53, -79.884)⟩. Each cell is 1km × 1km. We focus on signals from all DTV transmitters within a 150 mile radius of this area with estimated received powers higher than -130dBm. This results in a list of 37 DTV transmitters, of which we randomly pick 29 for building the classifier, and 8 for testing it. An illustration of the area, including the





Train Data - 29 Transmitters
Test Data - 8 Transmitters
Fraction of Cells
Received Signal Power 'P' (dBm)
P>-65
-65≥P>-85
-85≥P>-105
-105≥P>-114
-114≥P

Figure 4: (a) Transmitters in parts of Southwest Pennsylvania / East Ohio. (b) Distribution of received signal for the training and testing data in



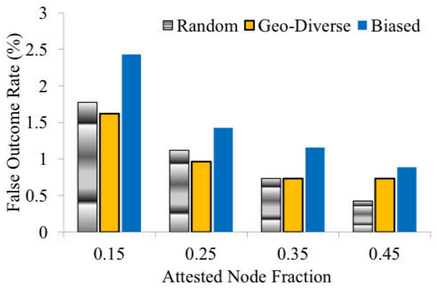


Figure 5: No attack; percentage of cells with ground truth average within the margin of error from the calculated aggregate (left) and false outcome rate (in percentage) as a function of the fraction of attested nodes (right).

location of the majority of DTV transmitters in provided in Figure 4(a). The distribution of the received signal power across all the cells in the region (from all 37 transmitters) is provided in Figure 4(b). Guided by approximate sample size requirements based on methods in Section 3.2, we consider nodes to be scattered with an expected density E_d of 50 nodes per cell. To add variation and randomness, we consider the number of nodes to be normally distributed with a mean of E_d, and a standard deviation of 10. Such densities will be easily achievable in urban areas. In suburban and rural areas, the densities need to be achieved through provisioning or other means in order for our approach to be fully effective.

4.1 No-Attack Performance

We first evaluate the accuracy of predictions generated by our approach when there is no attack. We compare the aggregate produced by our approach to the ground truth (real average power in the cell). In Figure 5(a) we show the percentage of cells for which the real average power is within the chosen margin of error $\epsilon = 3$dB from the calculated aggregate. The results show that our approach achieves a high overall success rate in terms of obtaining precise estimates of signal power in a region. They also show that despite Biased's weaker performance in some cases, in most cases the choice of inclusion strategy does not have a significant impact.

As a second performance metric in the absence of attacks, we introduce the *false outcome rate*, representing the fraction of unattacked cells with ground truth power above (below) the primary detection threshold of -114dBm that due to errors in our approach are mistakenly assigned an aggregate below (above) -114dBm. Figure 5(b) represents the false outcome rate as a function of the fraction of attested nodes. The results show that while overall false outcome rates are low, the Biased inclusion strategy is the weakest performer, particularly when the fraction of attested nodes is low. This can be explained by situations in which the few attested nodes are not providing values near the true average power in the cell, and the Biased inclusion strategy aggravates the situation by including similar data that effectively builds up on the already poor samples.

4.2 Performance against Attackers

To gauge performance in the presence of attacks, we simulate omniscient (and coordinated) attackers that perform exploitation and vandalism attacks. Attacker nodes act in cooperation and know the exact number, measurements, and type of all the other nodes, as well as the inclusion strategy in use (Random, Geo-diverse, or Biased). In cells where the ground truth is below the -114dBm threshold, they cooperate to perform exploitation to change the aggregate to a value above the threshold. Similarly, in cells where the ground truth is above -114dBm, they aim for vandalism by moving the aggregate to a value below the threshold. In both cases, the attackers minimize the deviation of their false reports from the measurements of un-compromised nodes by choosing to report values that move the aggregate slightly below (above) the threshold (.5 dB here) in order to perform exploitation (vandalism). This maximizes their chances of being included in the aggregate pool in Phase 1 and minimizes their chances of being detected in Phase 2. If the attackers conclude that the protections in Phase 1 do not allow them to 'flip' the aggregate, they refrain from reporting false reports to avoid detection.

To evaluate effectiveness against omniscient coordinated attacks, we introduce the *deterrence rate*. This metric represents the fraction of attacks by omniscient attackers that our approach thwarts. Deterrence may occur in phase 1 (by partial or total exclusion from the pool), or in phase 2 where their attack is detected by the classifier. We use data from 29 of the transmitters to build a unified classifier for the region [19] and test deterrence of attacks on the remaining 8 channels. The deterrence rates for cases with average attested fractions ranging from .15 to .35, and average attacker

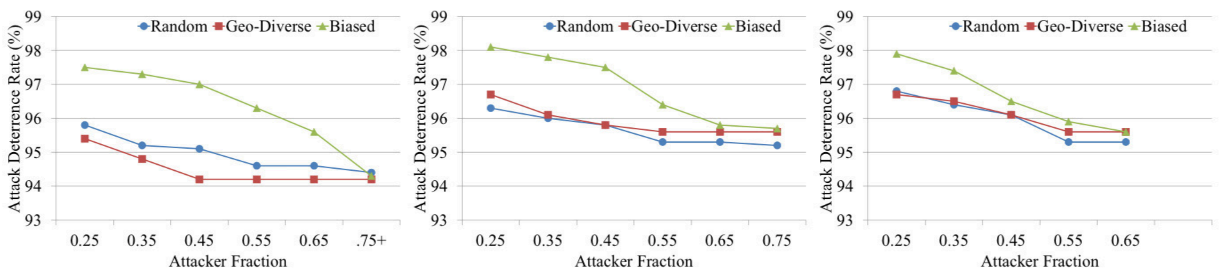


Figure 6: Attack deterrence rate (in percentage) when the average fraction of attested nodes is $.15$ (left), $.25$ (center), and $.35$ (right).

fraction ranging from .25 to .85 are presented in Figure 6. For attested fractions higher than .35, our results (omitted due to space constraints) show that it is more beneficial to avoid the complexities of our approach and only rely on the average of attested nodes.

In Figure 6, a surprising phenomenon can be seen in the case of Biased attacks. In some cases, when the attested fraction is increased (particularly from .25 to .35), the deterrence rate decreases. While this can be considered a flaw for the biased scheme, it can be described as follows. When the attested fraction is increased, there is less competition from regular un-compromised nodes (for attacker nodes) to report values close to the average of attested nodes and enter the aggregation pool. Therefore, the attackers have a higher chance of entering the pool with false reports, influencing the results, and passing Phase 1. The results in Figure 7 show this observation; unlike Random and Geo-diverse cases in which the deterrence at phase 1 does not change or increases as the attested fraction increases, the rate decreases for the Biased strategy.

Overall, the results show the following. (1) All three approaches are highly effective against omniscient attacks, even in cases where a small fraction of nodes are attested. (2) In terms of attack deterrence, the Biased inclusion strategy outperforms others. This is particularly true with lower attested and attacker fraction. This can be attributed to the difficulty of influencing the aggregate by attackers in these situations, since the attacker has to fulfil two conflicting goals of reporting values close to the attested average (to be included in pool) and at the same time far from the attested average (to move the aggregate and perform attack). (3) The relative outperformance of the Biased approach comes at the price of relatively higher false outcome rates when there is no attack.

5. ATTESTATION COSTS

Remote attestation can introduce potentially significant additional costs into a system. This section briefly surveys these costs for implementations of two remote attestation architectures. The first uses a TPM, which is a distinct coprocessor, whereas the second is implemented primarily in software, requiring only small hardware adaptations. The TPM-based architecture represents an upper bound on the cost of attestation, since the TPM is intended for use in desktop PCs with practically unlimited power supplies. The software-based architecture represents a low-cost alternative, although hardware and software innovations may result in architectures with even lower costs. The reason we include this section is to emphasize the fact that attestation introduces significant costs, which motivates our approach to leveraging relatively few attested nodes to establish trust in spectrum sensing results. The specific tradeoff between trust and cost can be made on a case-by-case basis.

Costs arise from various sources. Remote attestation support often requires additional hardware resources, which increase *manufacturing* costs. Some schemes involve a coprocessor, and even those primarily implemented in software may necessitate larger memories to store their code and data. Additional *energy* may be consumed by several components involved in a remote attestation transaction. Coprocessors and processors executing software routines both consume energy. Additionally, coprocessors usually consume some energy when inactive, and enlarged memories may require additional energy. Remote attestation transactions increase the amount of *network data* that is transmitted and received, which may also increase the energy consumption of the wireless radio. Increased network utilization can also impose *time* costs, as can remote attestation transaction processing.

We evaluated an Atmel AT97SC3203 TPM installed in a desktop PC. It imposes a manufacturing cost for the TPM chip itself, and potentially for expanded memories to support interface software installed on the attested processor. We measured its energy consumption using a Digital Multi-Meter (DMM). It draws 10.6mW of power when idle, which is likely to account for the bulk of its total energy consumption. It consumes around 58.9 mJ when an attestation certification is generated. Other operations require some energy, but are unlikely to contribute significantly to total consumption either due to their infrequent invocation or the fact that they do not involve expensive routines such as digital signature generation. Attestation operations require around 1.1 second to execute and generate at least 276 bytes of uncompressed data if the TPM uses a 2048-bit RSA key and the 160-bit SHA-1 hash algorithm, regardless of the specific protocol in use. For reference, we measured the energy consumption of a Digi XBee 802.15.4 radio using an oscilloscope, and determined that transmitting a packet with an x-byte payload consumed about $(0.017x + 1.83)$ mJ of energy at 1mW.

We also evaluated a software-based attestation scheme on an Atmel AVR32 AT32UC3A0512 microcontroller [25]. It only consumes extra energy when it is active. It uses Elliptic-Curve Cryptography (ECC) rather than RSA, which uses shorter keys (192 bits in this prototype) and simpler computations. Thus, although it does not use any hardware accelerators such as those in the TPM, it still consumes similar amounts of energy during attestation operations. Each operation takes about 0.6 seconds to execute. Due to the significantly shorter keys, each attestation operation only generates at least 68 bytes of data.

6. RELATED WORK

Much of the prior work in the context of white space networks uses various abnormality detection techniques to identify individual attackers within a cell as part of collaborative sensing. Such approaches, however, are not capable of detecting cells that form a majority in the cells [15, 31]. For example, Min *et al.*'s approach based on correlation-based filters fails to detect attackers

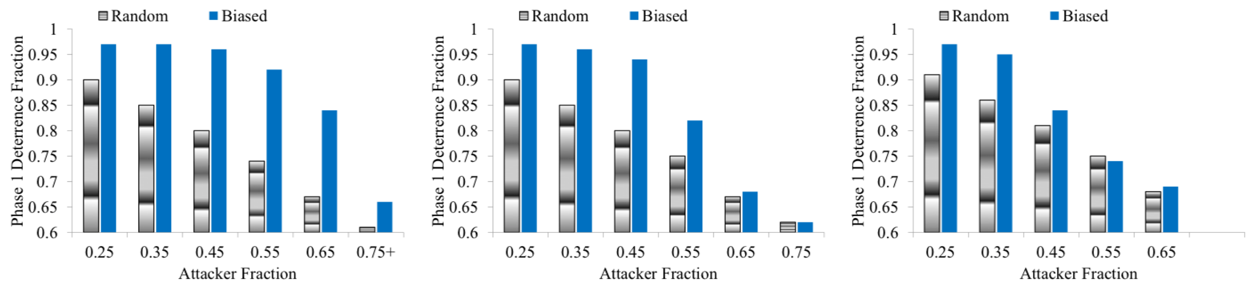


Figure 7: The fraction of attack deterrences in Phase 1. For each bar with value x, $1 - x$ is the fraction deterred in Phase 2. The average fraction of attested nodes is $.15$ (left), $.25$ (center), and $.35$ (right). Results for Geo-diverse (similar to Random) are omitted.

that constitute more than $1/3$ of the population of the nodes in a cell. Fatemieh *et al.* [18, 19] consider detecting attacker-dominated cells by outlier detection and classification techniques, however, their solutions do not consider remote attestation and fall short if a preponderance of neighboring cells are dominated by attackers.

Another body of related work in the context of white space networks considers primary user emulation (PUE) attacks [16, 29]. In a PUE, an attacker may modify the air interface of a radio to mimic a primary transmitter signal's characteristics, thereby causing legitimate secondary users to erroneously identify the attacker as a primary user. We consider this problem to be orthogonal to the problem we address.

In the context of sensor networks, Wagner introduced *resilient aggregation* [38], where he studies resilience of various aggregators to malicious nodes in an analytical framework based on statistical estimation theory and robust statistics. However, his work is limited to small regions and does not consider attack detection as we do. Zhang *et al.* [40] propose a framework that identifies readings not statistically consistent with the distribution of readings in a cluster of nearby sensors. Their proposal, however, is not able to handle situations where attacker can compromise a large fraction of the nodes in a cluster. Hur *et al.* [23] propose a trust-based framework in a grid in which each sensor builds trust values for neighbors and reports them to the local aggregator. Their solution, however, does not consider natural uncertainties in the data, does not provide a global view for a centralized aggregator, and cannot identify compromised 'regions.' For a survey on a closely related area of secure data aggregation in wireless sensor networks see [10].

There has been a number of works on utilizing remote attestation capability to achieve security in sensor networks. For example, there has been efforts on proposing architectures and building platforms [35], detecting compromised nodes [39], and other activities such as secure code update and key establishment [34]. To the best of our knowledge, no prior work has considered the problem of using attestation to defend against malicious false reports by omniscient attackers in the context of white-space distributed spectrum measurement.

Insider attacker detection in wireless networks is another area of related work. This problem has been explored in a general setting [13] as well as more specific contexts such as insider jammers. As an illustrative example in the general context of sensor networks, Liu *et al.* [28] propose a solution in which each node builds a distribution of the observed measurements around it and flags deviating neighbors as insider attackers. The solution, however, is local and peer to peer and does not work in areas with more than 25% attackers.

7. CONCLUSIONS

The use of statistical sequential estimation and classification methods can help evaluate and improve the trustworthiness of spectrum sensing results generated by a network containing a limited number of attested nodes. These methods reduce the total cost incurred by attestation. The results show that attestation capability for as low as 15% of the nodes can provide protection against more than 94% of the attacks from omniscient coordinated attackers. The protection improves as the fraction of attested nodes is increased. Our evaluation determined that the Biased node inclusion strategy is the most effective at deterring attacks, but also generates more false positives than Random or Geo-diverse strategies. These are not the only strategies that can be used, and future research should evaluate other strategies. One promising future direction is developing a framework for formulating costs associated with including regular and attested nodes, and systematically striking a balance between the costs (from spectrum data aggregation and remote attestation) and obtaining robust aggregation results.

8. ACKNOWLEDGEMENTS

We thank Farid Kianifard, Ranveer Chandra, and Ali Farhadi for their valuable comments. This work was supported in part by DOE DE-0000097, HHS 90TR0003-01, NSF CNS 09-64392, NASA 09-VVFCS1-09-0010, NSF CNS 09-17218, NSF CNS 07-16421, and grants from the MacArthur Foundation and Lockheed Martin. The views expressed are those of the authors only.

9. REFERENCES

[1] CogNea: Cognitive Networking Alliance. *http://www.cognea.org/*.

[2] FCC, ET Docket No FCC 08-260, November 2008.

[3] FCC, Second Memorandum Opinion and Order, ET Docket No FCC 10-174, September 2010.

[4] IEEE 802.22 WRAN WG on Broadband Wireless Access Standards. *http://www.ieee802.org/22*.

[5] Microsoft Research WhiteFi Service. *http://whitespaces.msresearch.us/*.

[6] S. 649: Radio Spectrum Inventory Act. *http://www.govtrack.us/congress/bill.xpd?bill=s111-649*.

[7] Trusted Computing Group. *http://www.trustedcomputinggroup.org/*.

[8] I. F. Akyildiz, W.-Y. Lee, M. C. Vuran, and S. Mohanty. Next generation/dynamic spectrum access/cognitive radio wireless networks: a survey. *Computer Networks*, 50(13):2127–2159, 2006.

[9] A. Algans, K. Pedersen, and P. Mogensen. Experimental analysis of the joint statistical properties of azimuth spread, delay spread, and shadow fading. *IEEE Journal on Selected Areas in Communications*, 20(3):523 –531, Apr. 2002.

[10] H. Alzaid, E. Foo, and J. G. Nieto. Secure data aggregation in wireless sensor network: a survey. *AISC '08: Proceedings of the sixth Australasian Conference on Information Security*, pages 93–105, 2008.

[11] P. Bahl, R. Chandra, T. Moscibroda, R. Murty, and M. Welsh. White space networking with wi-fi like connectivity. *Proceedings of the ACM SIGCOMM 2009 conference on Data communication*, pages 27–38, 2009.

[12] C. M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag New York, Inc., Secaucus, NJ, 2006.

[13] J. Branch, B. Szymanski, C. Giannella, R. Wolff, and H. Kargupta. In-network outlier detection in wireless sensor networks. *Distributed Computing Systems, 2006. ICDCS 2006. 26th IEEE International Conference on*, 2006.

[14] S. Capkun and J.-P. Hubaux. Secure positioning in wireless networks. *Selected Areas in Communications, IEEE Journal on*, 24(2):221 – 232, 2006.

[15] R. Chen, J.-M. Park, and K. Bian. Robust distributed spectrum sensing in cognitive radio networks. *INFOCOM 2008. The 27th Conference on Computer Communications. IEEE*, pages 1876 –1884, 2008.

[16] R. Chen, J.-M. Park, and J. Reed. Defense against primary user emulation attacks in cognitive radio networks. *IEEE Journal on Selected Areas in Communications*, 26(1):25–37, Jan. 2008.

[17] O. Fatemieh, R. Chandra, and C. A. Gunter. Low Cost and Secure Smart Meter Communications using the TV White Spaces. *Proceedings of ISRCS '10: IEEE International Symposium on Resilient Control Systems*, August. 2010.

[18] O. Fatemieh, R. Chandra, and C. A. Gunter. Secure Collaborative Sensing for Crowdsourcing Spectrum Data in White Space Networks. *Proceedings of DySPAN '10: IEEE International Dynamic Spectrum Access Networks Symposium*, April. 2010.

[19] O. Fatemieh, A. Farhadi, R. Chandra, and C. A. Gunter. Using Classification to Protect the Integrity of Spectrum Measurements in White Space Networks. *Proceedings of NDSS '11: 18th Annual Network and Distributed System Security Symposium*, Feb. 2011.

[20] M. Ghosh, N. Mukhopadhyay, and P. K. Sen. *Sequential Estimation*. John Wiley and Sons, Inc, New York, NY, 1997.

[21] M. Gudmundson. Correlation model for shadow fading in mobile radio systems. *Electronics Letters*, 27(23):2145 –2146, 1991.

[22] J. J. Higgins. *An Introduction to Modern Nonparametric Statistics*. Thomson Learning, Stamford, CT, 2004.

[23] J. Hur, Y. Lee, S.-M. Hong, and H. Yoon. Trust management for resilient wireless sensor networks. *Information Security and Cryptology - ICISC 2005*, pages 56–68, 2006.

[24] L. Lazos and R. Poovendran. Serloc: Robust localization for wireless sensor networks. *ACM Trans. Sen. Netw.*, 1(1):73–100, 2005.

[25] M. LeMay and C. Gunter. Cumulative attestation kernels for embedded systems. volume 5789 of *Lecture Notes in Computer Science*, pages 655–670. Springer Berlin / Heidelberg.

[26] Z. Li, W. Trappe, Y. Zhang, and B. Nath. Robust statistical methods for securing wireless localization in sensor networks. *IPSN '05: Proceedings of the 4th international symposium on Information processing in sensor networks*, page 12, 2005.

[27] D. Liu, P. Ning, A. Liu, C. Wang, and W. K. Du. Attack-resistant location estimation in wireless sensor networks. *ACM Transactions on Information and System Security (TISSEC)*, 11:22:1–22:39, July 2008.

[28] F. Liu, X. Cheng, and D. Chen. Insider attacker detection in wireless sensor networks. *INFOCOM 2007. 26th IEEE International Conference on Computer Communications. IEEE*, pages 1937 –1945, May 2007.

[29] Y. Liu, P. Ning, and H. Dai. Authenticating primary users' signals in cognitive radio networks via integrated cryptographic and wireless link signatures. *IEEE Symposium on Security and Privacy (Oakland)*, 2010.

[30] A. Min and K. Shin. An optimal sensing framework based on spatial rss-profile in cognitive radio networks. *Sensor, Mesh and Ad Hoc Communications and Networks, 2009. SECON '09. 6th Annual IEEE Communications Society Conference on*, pages 1–9, June 2009.

[31] A. Min, K. Shin, and X. Hu. Attack-tolerant distributed sensing for dynamic spectrum access networks. *ICNP '09: IEEE International Conference on Network Protocols*, pages 294–303, 2009.

[32] J. Newsome, E. Shi, D. Song, and A. Perrig. The sybil attack in sensor networks: analysis & defenses. *IPSN '04: Proceedings of the 3rd international symposium on Information processing in sensor networks*, 2004.

[33] T. Rappaport. *Wireless Communications: Principles and Practice*. IEEE Press, New York, 1996.

[34] A. Seshadri, M. Luk, and A. Perrig. SAKE: Software attestation for key establishment in sensor networks. *Proceedings of the 2008 International Conference on Distributed Computing in Sensor Systems (DCOSS)*, June 2008.

[35] A. Seshadri, A. Perrig, L. van Doorn, and P. Khosla. SWATT: SoftWare-based ATTestation for embedded devices. *IEEE Symposium on Security and Privacy (Oakland)*, May 2004.

[36] C. R. Stevenson, G. Chouinard, Z. Lei, W. Hu, S. J. Shellhammer, and W. Caldwell. Ieee 802.22: the first cognitive radio wireless regional area network standard. *Communications Magazine*, 47(1):130–138, 2009.

[37] R. Tandra, A. Sahai, and S. Mishra. What is a spectrum hole and what does it take to recognize one? *IEEE Magazine Special Issue on Cognitive Radio*, 97(5):824–848, May 2009.

[38] D. Wagner. Resilient aggregation in sensor networks. *SASN '04: Proceedings of the 2nd ACM workshop on Security of ad hoc and sensor networks*, pages 78–87, 2004.

[39] Y. Yang, X. Wang, S. Zhu, and G. Cao. Distributed software-based attestation for node compromise detection in sensor networks. *Reliable Distributed Systems, 2007. SRDS 2007. 26th IEEE International Symposium on*, pages 219 –230, 2007.

[40] W. Zhang, S. Das, and Y. Liu. A trust based framework for secure data aggregation in wireless sensor networks. *Sensor and Ad Hoc Communications and Networks, 2006. SECON '06. 2006 3rd Annual IEEE Communications Society on*, 1:60–69, Sept. 2006.

Don't Bump, Shake on It: The Exploitation of a Popular Accelerometer-Based Smart Phone Exchange and Its Secure Replacement

Ahren Studer
Carnegie Mellon University
astuder@cmu.edu

Timothy Passaro
Carnegie Mellon University
tpassaro@andrew.cmu.edu

Lujo Bauer
Carnegie Mellon University
lbauer@cmu.edu

Abstract

As the capabilities of smartphones increase, users are beginning to rely on these mobile and ubiquitous platforms to perform more tasks. In addition to traditional computing tasks, people are beginning to use smartphones to interact with people they meet. Often this interaction begins with an exchange, e.g., of cryptographic keys. Hence, a number of protocols have been developed to facilitate this exchange. Unfortunately, those protocols that provide strong security guarantees often suffer from usability problems, and easy-to-use protocols may lack the desired security guarantees.

In this work, we highlight the danger of relying on usable-but-perhaps-not-secure protocols by demonstrating an easy-to-carry-out man-in-the-middle attack against Bump, the most popular exchange protocol for smartphones. We then present Shake on It (Shot), a new exchange protocol that is both usable and provides strong security properties. In Shot, the phones use vibrators and accelerometers to exchange information in a fashion that demonstratively identifies to the users that the two phones in physical contact are communicating. The vibrated information allows the phones to authenticate subsequent messages, which are exchanged using a server. Our implementation of Shot on DROID smartphones demonstrates that Shot can provide a secure exchange with a similar level of execution time and user effort as Bump.

1. INTRODUCTION

As the functionality and computing power of smartphones increase, users are leveraging these devices to perform more of their computing tasks. In addition to traditional tasks such as email, gaming, banking, and maintaining a schedule, the mobility and ubiquity of smartphones allows people to use these devices to establish ad hoc associations with people they meet. For example, people may exchange phone numbers, email addresses, and social network identities or even use their phones to enable the exchange of funds via an online service (e.g., PayPal). During these exchanges, phones typically use the wireless channel to perform the majority of the communication. The wireless channel makes exchanges easier for users since they do not have to carry cables to connect the phones. However, users are unable to observe the endpoints of wireless communication and without a secure protocol a malicious party can insert themselves into the exchange as part of a man-in-the-middle (MitM) attack.

Exchange protocols for smartphones require varying levels of user involvement and provide different security guarantees. The widely deployed Bump [3] and Bluetooth pairing [2] protocols represent distant points in the spectrum of user involvement and security guarantees. Bump requires users to perform only a simple gesture, but, as we demonstrate, is vulnerable to MitM attacks under realistic conditions. Bluetooth pairing [2] is resistant to such attacks, but requires significant user involvement to correctly compare a checksum across phones.

In the first part of this paper, we highlight the danger of relying on usable-but-perhaps-not-secure protocols by demonstrating a MitM attack against Bump. Bump is the most popular exchange protocol for smartphones[1] and claims to provide an excellent user experience while ensuring security [3]. Due to the simplicity of the operation, Bump is used in a number of applications. For example, PayPal's mobile app uses Bump to exchange account information and send money to nearby friends [20]. Bump's security is based on the Bump server's ability to determine what phones are physically interacting based on the time, location, and the force with which the two phones were physically bumped together. Bump provides a very nice user experience, but is vulnerable to attack. An attacker may be able to observe when and where users bump their phones together and estimate the force of the bump. Without any secret information to identify the pair of phones to the server, an attacker can submit similar information about a bump to the server. When presented with similar information, the server may transfer data between the wrong phones. We demonstrate how an attacker under realistic conditions can use this approach to launch a MitM attack against Bump. Other works have also suggested using accelerometers or vibrators and accelerometers to facilitate an exchange of information between phones [6, 9, 10, 13, 17, 23]. However, our attack on Bump and prior work [8] demonstrate that a phys-

[1]Popularity is based on data from `http://techcrunch.com/2011/01/19/iphone-ipad-top-app-downloads/` and `http://www.androidapps.com/` which use download count to determine popularity. According to the Bump blog, over 25 million users have installed Bump.

ically present attacker may be able to violate the secrecy of this channel, requiring a new approach to ensure security.

In the second part of this work, we demonstrate how phones can leverage accelerometer readings to assist in the secure exchange of information while maintaining the limited user involvement offered by Bump. We propose Shake on It (Shot), a protocol designed specifically for smartphones that requires little user interaction. The novel idea is to use phones' vibration function and accelerometers as an authentic, but not secret, human-observable communication channel. The phones leverage data exchanged on this channel to verify data exchanged over the wireless channel. Only when the two phones are in physical contact can they communicate via the vibration-to-accelerometer channel, providing demonstrative identification to the users of the devices exchanging information [1]. If a remote party (one not in physical contact) tries to inject information on the channel, users will notice the additional vibrations (a potential attack) and stop the exchange. Since attackers can eavesdrop on this channel [8] and control the wireless channel, the phones use data from the vibrated exchange and cryptographic operations to authenticate subsequent exchanges over the wireless channel, without users having to compare any information. When two people run Shot, they have to perform only three simple tasks: select what data is to be exchanged, hold the phones together until the phones beep to indicate completion, and cancel the exchange if they feel vibrations from devices other than the two phones. We argue that Shot occupies a new point on the spectrum of exchange protocols, providing greater security than comparably convenient protocols, and greater convenience than comparably secure protocols.

We have implemented Shot on a commodity smartphone and evaluated the performance of the protocol. Our analysis and evaluation show that Shot provides a secure exchange while requiring the same level of user effort and execution time as the popular Bump protocol. When evaluated on the DROID smartphone, Shot was able to reliably complete an exchange in 15.8 seconds, on average.

In summary, this paper offers the following contributions: (1) demonstration of an attack on Bump under realistic conditions; (2) a description of the Shot exchange protocol for smartphones, which leverages an authentic, but not secret, vibrator-to-accelerometer channel; and (3) an implementation of the Shot protocol on a smartphone.

2. PROBLEM DEFINITION

The goal of this work is to provide two people (users A and B) who meet in person a user-friendly mechanism that allows the authentic exchange of information (A's information I_A and B's information I_B) using their phones (P_A and P_B). After the exchange is complete, P_A will have received I_B and P_B will have received I_A, or P_A and P_B will both detect with high probability that an error has occurred and discard the information.

A user-friendly solution should require limited user involvement and should complete execution in a reasonable amount of time. The exchange should not, e.g., require users to type several bytes worth of information into either phone or compare a checksum (e.g., a string of hex digits [2,12,26], a series of words [7], or a graphical image [14,21]). User studies have shown that a redesigned interface [25] or the comparison of words or images [11] reduce the number of errors, but still require non-negligible user involvement. The desired level of user involvement is a simple action, such as a physical gesture, that indicates to one phone which other phone should be involved in the exchange. Solutions exist that require the user to take a photograph of the other phone [18], shake the two phones together [6,10,13,17], or simply point the phones at each other [1], but each solution has drawbacks that negatively impact security or operation (see Section 9 for a discussion of related work). Given the wide acceptance of Bump, we consider the execution time of a Bump exchange as a reasonable amount of time. Depending on the platform, Bump takes between 9.4 and 37.8 seconds. Section 8 has more details on the execution time of Bump.

During an exchange, a malicious party M may attempt to inject its own information (I_M) or other data into the exchange such that P_A or P_B accept something other than I_B or I_A, respectively. A secure exchange allows P_A and P_B to exchange I_A and I_B or detect the insertion of any other information into the exchange with a high probability.

After discussing our assumptions, we give a detailed description of our attacker model.

2.1 Assumptions

In this work, we assume smartphones are equipped with the hardware, software, and connectivity needed to execute a Bump or Shot exchange.

Current smartphones are equipped with a vibrator and an accelerometer to provide silent notifications to the owner and to allow correct orientation of an image on a rotatable screen. Smartphones also allow the installation of generic software that can access the vibrator and accelerometer functionality, and have Internet connectivity the majority of the time via the cellular network or WiFi.

2.2 Attacker Model

An attacker's goal during the smartphone exchange between phones P_A and P_B is to convince P_A or P_B to accept information other than I_A and I_B. We consider an attacker that may be in the same room as A and B, knows the value of I_A and I_B, and has bounded computational capabilities. We also assume any software on P_A and P_B is outside of the attacker's control.

We assume the attacker has control over the wireless channel between the two phones and is able to intercept, modify, delay, or inject messages. However, the attacker does not control human-observable channels between the two phones (i.e., the visual or vibration channel). The attacker can accurately eavesdrop on these channels, but users can detect the attacker's attempts to insert information on a human-observable channel (i.e., insertion of a third phone into the visual channel or shaking the phones to inject data onto the vibration channel).

We assume an attacker knows a priori what information the two users want to exchange. Since most users are exchanging contact information (e.g., phone numbers or online account IDs), this information can often be found online.

It is infeasible for a computationally bounded attacker to break various properties of different cryptographic primitives of the appropriate strength. Hence, hash functions with sufficiently long outputs are one-way and second preimage resistant (i.e., given a hash function $h()$ and a hash output $y = h(x)$, an attacker is unable to find x or another value x' such that $h(x') = y$). We also assume digital signatures are secure against selective forgery. This means that without knowledge of the private key, it is infeasible for the attacker to create a signature (σ) for an attacker-selected message m such that the corresponding public key verifies the signature message pair (σ, m).

3. Bump EXCHANGE PROTOCOL

In this section, we begin by describing the Bump exchange. We then explain why the exchange is vulnerable and provide an analysis of our attack.

3.1 The Bump Protocol

The Bump exchange is meant to allow two phones (P_A and P_B) to exchange information. During an exchange, the users physically bump P_A and P_B together; each phone sends the time, location, and force of the bump and the information it wants to exchange to the bump server; the server uses the information about the bumps to determine which phones want to exchange information and returns to each phone the other phone's ID; and the users decide to complete the exchange based on the ID. Based on information provided on Bump Technology's webpage (`http://www.bu.mp`) and analysis of packets sent during a Bump exchange, we were able to determine approximately how the Bump protocol works. Figure 1 provides our understanding of the operation of Bump for a single phone involved in a Bump exchange.

3.2 Vulnerabilities in Bump and Their Exploitation

Bump is insecure due to three aspects: inaccuracies in the measurement of the physical aspects of the bump, the ability of a sender to control the ID used during confirmation, and server flexibility in accepting delayed bump requests. Because of these issues, an attacker is able to successfully launch a MitM attack.

Sensor Inaccuracies. Given limited sensor accuracy, the phone is unable to know its exact location or how hard the phone was bumped. Given inaccuracies in different phones' clocks, the server is unable to know exactly when a bump occurred. Without accurate information, the server must use approximate matching, which can associate bumps that occurred several meters apart, with different forces, and at slightly different times. We were able to confirm that the server will match bumps that differed in each of these three aspects. This allows an attacker to block P_B's bump request and have the server match an attacker's request with P_A. However, a full MitM attack is not achieved because both the confirmation question ("Connect with *ID*?") and the failure of P_B's request allow users to detect the attack.

Spoofable ID. Circumventing the confirmation question so that a MitM attack goes undetected in Bump is simple. Since a participant controls the ID sent to the other phone during an exchange, the attacker can choose the same ID as the victim it is impersonating, but send it with different accompanying data (e.g., the attacker's email or public key).

Acceptance of Delayed Requests. Different networks can cause different delays, such that the request from P_A arrives before the request from P_B, which arrives long after P_B claims to have bumped. Presumably because of such potential delays, if no matching database entry exists, the server will wait some period of time before responding to P_A's request, and will consider P_B's request valid even if it arrives substantially after its claimed bump time. At the same time, the server will consider two requests a match even if they report somewhat different times and bump values. In short, Bump accepts both stale and approximate requests.

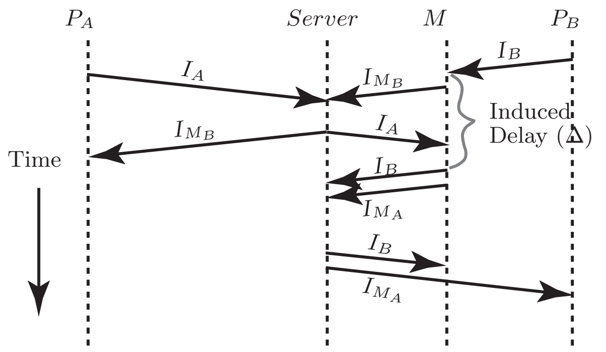


Figure 2: Timing of packet exchanges during the attack on Bump (accelerometer and location information omitted for clarity).

The Attack. An attacker with the ability to submit similar bumps to the server, spoof legitimate users' IDs, and delay legitimate requests can successfully complete the following MitM attack against Bump. Figure 2 sketches the attack. After A and B bump phones, the attacker delays P_B's request containing I_B and sends its own request impersonating P_B with information I_{M_B} and similar bump values. The server associates the attacker's request with P_A's request, since no other similar entries exist in the database. Next, the attacker forwards P_B's delayed request and sends a request impersonating P_A with information I_{M_A} and the same bump values. In response, the server associates those two requests. Delay Δ, the duration by which the attacker delays the server's receipt of P_B's request, is an important parameter in the attack. If Δ is too short, the server will correctly associate the legitimate users' bumps or claim all of the bumps were too similar and ask the phones to bump again. If Δ is too large ($>$ 3 seconds), the server returns an error in response to the delayed request.

In the next subsection, we describe how such an attack can happen in practice and empirically investigate the probability of a successful attack based on different values of Δ.

3.3 Attack Implementation and Analysis

To successfully perform a man-in-the-middle attack against Bump, an attacker must be able to observe the bump and delay packets. A maliciously controlled access point (AP) provides control of packets. In our attack setup, we use a MacBook Pro with OS X 10.5 and Dummynet [5] as an AP. The laptop is connected to the Internet via the ethernet port and can forward packets from the WiFi network to the Internet. In a real attack, the attacker could trick users into associating with the attacker's AP by selecting a common SSID (e.g., "linksys"). By default, a phone will use a WiFi network with a known SSID, rather than the cellular network.

To evaluate the attack, we used real phones with three different people to bump the two sets of phones. We used

Initialization:		
1. P_A	: $loc \leftarrow findLoc()$	The phone determines its current location via GPS or WiFi information.
2. $P_A \overset{Internet}{\longleftrightarrow} S$	: establish TLS	The phone establishes a TLS connection with the server.
Exchange:		
3. $A \overset{acc.}{\rightsquigarrow} P_A$	: a_{bump}, t	Bumping the phone induces an accelerometer reading a_{bump} at t.
4. $P_A \xrightarrow{TLS} S$	: a_{bump}, t, loc, I_A	The phone sends the acceleration, time, and location of the bump and the user's info to the server.
5. S	: $I_{match} = match(a_{bump}, t, loc)$ $resp = I_{match}.ID$	The server checks its database of recent bump requests for similar bumps, and returns the ID field from the result. $resp = \emptyset$ when no match is found. $resp =$ **"Bump Again"** when > 1 match is found.
6. $S \xrightarrow{TLS} P_A$	: $resp$	The server returns the ID of the match (or the error code).
7. $P_A \xrightarrow{Screen} A$	: If $((resp \neq \emptyset)$ and $(resp \neq$ **"Bump Again"**$))$ **" Connect with** ***resp*****? Yes/No"**	After a successful match, the phone asks the user if they want to connect with the other user. Otherwise, the user is presented an error message.
8. The phone sends the user's selection to the server. If both phones return "yes", the server returns the other phone's data. If either phone returns "no", the server tells the other phone the exchange was cancelled.		

Figure 1: Operations during a Bump exchange where user A uses phone P_A to exchange information I_A using the server S.

four iPhone[2] 3G smartphones running iOS version 4.2.1 and Bump 2.4.0. Two humans bumped phones P_A and P_B together while another human played the role of the attacker and tried to bump phones P_{M_A} and P_{M_B} together at roughly the same time and with the same force. This was meant to simulate an attacker observing victims across the room using Bump to exchange phone numbers at a bar or exchange money to reimburse one user for the other user's share of a tab. Dummynet was configured to delay P_B's and P_{M_A}'s requests by Δ as part of the attack (see Figure 2).

To evaluate Bump, we varied the induced delay (Δ) from 0 to 3 seconds in increments of 0.5 seconds and ran 10 exchanges for each setting. During an exchange, a phone can experience one of three potential outcomes:

SE Successful Exchange: The server returns a potential match and asks if the user wants to communicate with that ID.

BA Bump Again: The server finds multiple similar entries and asks the phone to bump again.

OO Only One: The server is unable to find a similar entry in the database. During analysis, we found that this is also returned if the induced delay is large.

Based on the individual phones' outcomes, we classify the result of the attack as one of the following:

- **Successful Exchange:** P_A receives B's information (**SE**) and P_B receives A's information (**SE**).
- **Successful Attack:** P_A receives the impersonation of B (I_{M_B}) and P_B receives the impersonation of A (I_{M_A}). Both phones have outcome **SE**, but receive attacker's information.
- **Detectable Attack:** P_A receives the impersonation of B (I_{M_B}), but P_B receives bump again or only one, i.e., **SE** for A and **BA** or **OO** for B, depending on whether Δ was too small or too large.
- **Other:** P_A and P_B receive Bump Again or Only One. Neither phone experiences outcome **SE**.

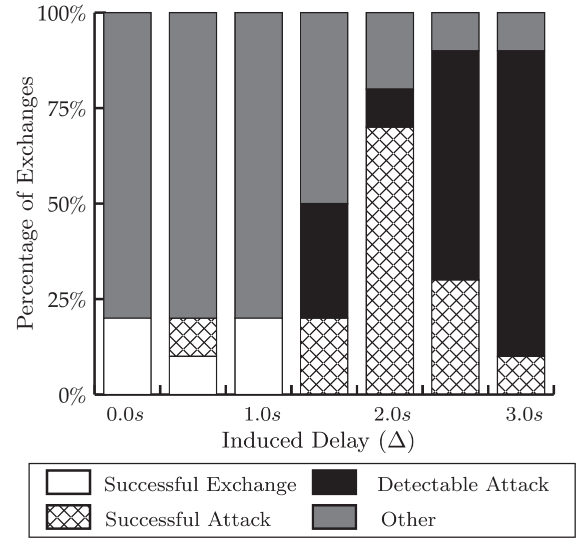


Figure 3: Attack Results on Bump with Varying Values of Δ

Our results are summarized in Figure 3 and show that a MitM attack against Bump is feasible. We found the optimal induced delay (Δ) to be around 2 seconds. With that delay, 70% of the attacks succeeded. With less delay, the server detects multiple similar requests and returns "Bump Again".

[2]We used iPhones for our attack because at the time of our analysis of Bump in February of 2011 iOS supported a more recent and reliable version of Bump, version 2.4.0, while phones with Android were limited to version 1.3.2.

With more delay, the server detects the long delay for P_B's request and instead of associating the request with I_{M_A}, returns an "Only One" response. For delays longer than 3 seconds, the attack is detectable; the server consistently pairs I_A and I_{M_B}, but returns "Only One" in response to I_B and I_{M_A}.

4. EXCHANGE GUIDELINES

This section discusses some of the lessons learned from and problems encountered in Bump. Based on these, we developed the Shake on It (Shot) exchange, described in Section 5.

Server-based Communication: Smartphones are connected to the Internet the majority of the time via the cellular network or WiFi. By using a server to exchange information, Bump circumvents some challenges associated with trying to establish local communication. For example, iPhones allow Bluetooth connections only to other iPhones or computers.[3] Ad hoc communication over WiFi is another option, but requires users of Android-based phones to subvert the OS to enable phone-to-phone WiFi broadcast.[4] A server allows smartphones from different vendors to communicate quickly and easily, without requiring the owner to modify the operating system.

One drawback to using a server to communicate is that the server needs to know which phones are trying to exchange data. With wireless communication, the phones can assume any broadcast data came from the other phone. Since all phones using Shot share the same server, the server needs a way to differentiate each pair of phones. As such, before the phones communicate via the server, the phones must agree on a value we call the "pair identifier," which allows the server to route traffic from one phone to the other.

Accelerometers as an Authentic Channel: A number of works use a bump, shake, or gesture to intuitively indicate which phones are to exchange data [3,6,10,13,17]. The accelerometer provides a way to convert physical interaction into a label to identify potential endpoints to a server or to derive a secret used to detect a MitM attack. This human-observable communication also provides the users demonstrative identification of the endpoints of the exchange.

Unfortunately, a physically present attacker can observe the movements and leverage real-time motion tracking [15] and high speed cameras to quantify the accelerations during the bump, shake, or gesture. Once the accelerations are known, the information the phones share lacks the secrecy needed to secure communication. Rather than assuming secrecy, we need a protocol that can provide security and usability with only authenticity.

Attack Detection on the Phones: Since attackers may be able to observe any information received by the accelerometer, the phones are unable to present any information to the server that allows the server to isolate the correct endpoints in an exchange. Without such a mechanism, the phones may receive the wrong information, but are unable to detect the error. Increasing user involvement by comparing the received data would solve the problem, but reduces usability. Instead, we want a protocol that allows the phones to detect reception of the wrong information from the server, while limiting user involvement to something as simple as putting the phones together.

Based on these observations, we need a protocol that begins by establishing a pair identifier, but also provides an easy way to verify authenticity. The phones can use the pair identifier to exchange information using the server, and use authentic information to verify the correct phones and data were involved in the exchange, without requiring secrecy or involving users in the verification.

5. THE Shake on It (Shot) EXCHANGE

In this section, we describe Shot, a secure exchange protocol for smartphones that provides a user experience similar to Bump. We leverage the smartphones' vibration function and accelerometers to transmit an authentic phone-selected message from one phone to the other. This allows the phones to bootstrap communicate via an untrusted server and verify received data without involving the user. In Section 5.1 we explain certain design decisions and provide an overview of the protocol. Section 5.2 describes Shot in detail, and we conclude this section with a security analysis.

The version of Shot presented here only ensures the integrity of the exchanged information. If secrecy is desired, the two parties can use Shot to authentically exchange public keys, and use those keys to establish a shared symmetric key for encryption of information.

5.1 Shot Overview

Based on the guidelines in Section 4, we want a protocol that takes advantage of smartphones' accelerometers for demonstrative identification and authentic communication. For Shot, we take this one step further and leverage the smartphone's vibration function to send a phone-selected message to the other phone's accelerometer. With this capability, the naive approach would be to have users hold phones P_A and P_B together. Once together, P_A would vibrate I_A to P_B and P_B would vibrate I_B to P_A. Unfortunately, the vibration-to-accelerometer channel is too slow for this naive approach to be user friendly (see Section 7).

Several works explain how two parties can exchange data over an insecure medium (e.g., a server) and use an authentic channel (e.g., human comparison or vibration) to verify the exchange, using a checksum derived from the exchanged data [2,7,12,21,26]. However, these protocols require the phones to exchange data before they can calculate the checksum. If using a server to communicate, the phones need a pair identifier to communicate via the server, before the phones use the vibration channel to exchange the checksum. A protocol could use two long vibrations (one at the beginning to exchange a pair identifier and another to exchange the checksum), but vibrating all of that information would be slow. Users could copy a pair identifier from one phone to the other, but that is cumbersome.

Shot starts by using the vibration channel to send a message that performs two functions: 1) acts as a pair identifier and 2) acts as a pre-authenticator to verify the authenticity of data exchanged from one phone to the other. Much like Talking to Strangers [1] and Seeing-Is-Believing (SiB) [18], Shot begins with the exchange of a pre-authenticator. However, Shot provides demonstrative identification and secure exchange with a single pre-authenticator. After P_A vibrates the pair identifier, the phones exchange data using the server

[3] `http://discussions.apple.com/thread.jspa?threadID=1460770`

[4] `http://code.google.com/p/android/issues/detail?id=82`

and confirm the exchange in the following fashion:

1. P_B uses the pre-authenticator/pair ID to verify that a public key obtained from the server belongs to the phone pressed against P_B.

2. P_A digitally signs I_A and a copy of the other phone's information, I_B, which P_A received from the server.

3. P_B verifies the signature from the server is a valid signature over the other phone's information and its own information (I_A and I_B). If the signature is correct, P_B vibrates back a positive response to indicate that the other phone signed a copy of the information P_B received and sent.

4. P_A uses the vibrated response to determine whether the protocol completed successfully.

Shot uses two vibrated messages, since one vibration in each direction is needed to achieve demonstrative identification for both phones. The second vibration is a yes/no response that can be quickly transmitted in order to maintain a short execution time.

5.2 Shot Exchange

Shot provides a user-friendly way to exchange information between smartphones by leveraging the authentic nature of the vibration-to-accelerometer channel, communication via a server, and the phones' ample computational capabilities. To minimize computation and communication over the vibration channel each phone has a role. For the remainder of this work, rather than P_A and P_B, we call one phone the Endorser (P_E) and the other phone the Verifier (P_V). After users select what data to exchange and phones' roles are agreed assigned, Shot consists of 4 phases: 1) exchange of the pair identifier, 2) exchange of data via the server, 3) signing the data, and 4) confirmation of the data. Figure 4 shows the steps associated with the Shot exchange.

① **Exchange of the Pair Identifier** During the initial phase, the phones exchange a pair identifier which bootstraps communication through the server and allows P_V to verify P_E's public key ($K^+_{P_E}$) later in the protocol. When the protocol begins, P_E calculates a shortened hash of its public key (Step 1). We limit the hash to 80 bits to balance security and time needed to transmit the value over the vibration channel. After the users physically place their phones together, P_E vibrates this hash (Step 2). This vibration demonstrates to P_V that the other phone in the exchange has a public key that hashes to this value. Since this hash is in practice unique, we can also use it as the pair identifier to establish communication through the server.

② **Exchange via the Server** Once the phones know how to identify the pair to the server, the phones use the server to exchange the Endorser's public key and any other information to be exchanged (P_E's info I_E and P_V's info I_V).

During each transmission (Steps 3 & 4) and retrieval (Steps 5 & 6), the phone begins the connection by sending the pair identifier (h). The server uses h to know how to record and retrieve the information associated with a pair of phones. If at any point a phone receives the wrong information from the server (e.g., P_V finds that $I_V \neq I_V'$ or $h' \neq h$), the phone assumes an attack has occurred and aborts the protocol.

③ **Signing the Data** Once I_E and I_V have been exchanged, the phones start verifying the authenticity of the data. P_E uses its private key ($K^-_{P_E}$) to sign the data it believes was exchanged so that P_V can detect whether the information was modified in transit. Specifically, P_E signs the concatenation of its own information and the potential copy of the Verifier's information (Step 7). P_E then uses the server to send the signature to P_V (Steps 8 & 9).

④ **Confirmation of the Data** The final phase has three checks to detect if the exchange was successful:
• P_V verifies $K^+_{P_E}$ belongs to the phone P_V is pressed against.
• P_V verifies that the owner of the authenticated public key received I_V and sent the information received in Step 5.
• P_E verifies that the phone it is pressed against received a valid signature, confirming that the data P_E signed in Step 6 is what each phone sent and received during the exchange.

```
① Exchange of the Pair Identifier:
1.  $P_E$                                   : $h = Hash(K^+_{P_E})$
2.  $P_E \overset{acc.}{\rightsquigarrow} P_V$ : $h$
② Exchange Via the Server
3.  $P_E \rightarrow S$                     : $h, K^+_{P_E}, I_E$
4.  $P_V \rightarrow S$                     : $h, I_V$
5.  $S \rightarrow P_V$                     : $h', K^+_{P_E}{}', I_E', I_V'$
6.  $S \rightarrow P_E$                     : $h', K^+_{P_E}{}', I_E', I_V'$
③ Signing the Data:
7.  $P_E$                                   : $\sigma = Sign(K^-_{P_E}, I_E || I_V')$
8.  $P_E \rightarrow S$                     : $h, \sigma$
9.  $S \rightarrow P_V$                     : $h', \sigma'$
④ Confirmation of the Data:
10. $P_V$                                   : if($h == Hash(K^+_{P_E}{}')$ and
11.                                             $Verify(\sigma', K^+_{P_E}{}', I_E' || I_V)$)
                                                  $result$ = "yes"
                                                  save($I_E'$)
                                              else $result$ = "no"
12. $P_V \overset{acc.}{\rightsquigarrow} P_E$ : $result$
13. $P_E$                                   : if($result$ == "yes")
                                                  save($I_V'$)
14. $P_s, P_V$                              : Sound Tone
```

Figure 4: Shot exchange between P_E and P_V utilizing the server S. (X' is used to indicate a potentially modified value of X that has been transfered over the attacker control wireless medium.)

This verification process begins by P_V verifying that the hash of the received public key matches the hash from the vibration channel (Step 10). This checks whether the public key P_V received belongs to the phone physically pressed against P_V. After receiving the other phone's public key, P_V verifies the signature from the server (Step 11). If the public key and signature are correct, P_V knows the data it received came from the phone it is pressed against and that that phone received I_V. P_V uses the vibration channel to send a short confirmation to P_E (Step 12). This vibration informs P_E whether the phone it is pressed against received a valid signature (verified using P_E's public key) over the exchanged data. Finally, the phones sound a tone to indicate that the protocol has completed and users can stop holding the phones together.

In the next section we describe why Shot is secure provided some properties of the underlying cryptographic primitives and authenticity of the vibration channel hold.

6. SECURITY ANALYSIS OF Shot

For Shot to securely exchange information the following four properties are necessary:

- The pre-authenticator received by P_V was sent by P_E.
- P_V can detect if it received a copy of $K^+_{P_E}$ or the wrong public key.
- P_E is the only entity that can generate a signature which $K^+_{P_E}$ verifies.
- Only P_V can indicate to P_E whether the signature it received verifies the information P_V sent and received.

We discuss the properties needed from different cryptographic primitives to fulfill the second and third properties before discussing the authenticity of the vibration channel which is needed to fulfill the first and last properties.

6.1 Cryptographic Primitives

If an attacker is unable to inject information onto the vibration channel (see next subsection), a hash function that is second-preimage resistant allows P_V to verify it received a copy of P_E's public key. Given an authentic copy of P_E's public key, a signing algorithm that is secure against selective forgery allows P_V to verify P_E received P_V's information and sent the information P_V received.

If an attacker wants P_V to accept a different public key, the attacker has to find a different public key ($K^+_{P_M}$) such that the truncated hashes are the same (i.e., $Hash(K^+_{P_E}) = Hash(K^+_{P_M})$). However, if the hash function is second-preimage resistant, it is infeasible for an attacker to find such a public key, even if the hash is truncated to 80-bits [22].

Without a way to convince P_V to accept a different public key, an attacker needs to produce a signature over incorrect exchange information that will verify with key $K^+_{P_E}$. If the authentic public key were to verify an attacker-generated signature for the message $X||I_V$, P_V would believe the other phone signed that message, indicating P_E sent X instead of I_E. However, if the signature scheme used is secure against selective forgery, it is infeasible for an attacker to produce such a signature. Without a valid signature, P_V will reject the information and thwart the attack.

Provided the hash function and the signature scheme are secure, the phones will detect any type of active attack against data exchanged over the wireless channel. Once the attack is detected, the phones will discard the information. This fail safe operation does mean an attacker can launch a Denial-of-Service (DoS) attack. However, DoS attacks are outside of the scope of this work since the attacker could also jam the wireless channel to prevent communication.

6.2 Vibration as an Authentic Channel

For Shot to be secure, only P_E and P_V should be able to send information on the vibration-to-accelerometer channel. However, if an attacker were to send information on the channel, a user could detect the vibrations and abort the exchange. Without a way for attackers to send messages on the channel, only P_E is able to send a pre-authenticator and only P_V is able to send confirmation of a valid signature.

Users can detect when other parties are trying to send information on the vibration-to-accelerometer channel. The two users know what phones are exchanging information, and will detect if another device is vibrating against the phones. Holding two phones together is an intuitive way to indicate which phones should exchange information and should experience a low rate of operator error. Instead, we have to worry about the attacker remotely inducing vibrations. However, unless the attacker can focus those vibrations precisely on the phones, the user(s) holding the phones will notice the additional vibrations and stop the exchange. For example, consider an attacker that produces a loud tone at a low frequency in an attempt to vibrate the two phones remotely. The tone needs to be quite loud to induce vibrations which are comparable to the vibrations from another phone in direct physical contact.[5] Even if the frequency of the tone is below the human audible range, the users' will feel the vibrations and abort the exchange.

Without a way to break the security properties provided by the underlying cryptographic primitives, an attacker must find a way to inject a message on the vibration-to-accelerometer channel to successfully subvert a Shot exchange. However, users are likely to detect attempts to remotely induce vibrations, thwarting the attack.

7. Shot IMPLEMENTATION

In this section, we describe our implementation of Shot for the Motorola DROID smartphone.

7.1 Shot for Android

Our implementation of Shot was written for and tested on Motorola DROID smartphones with Android version 2.2. However, the system can be ported to any mobile phone with a vibration function and accelerometer. If installed on other phones, effective bit rates for the vibrator-to-accelerometer may change with different hardware based on access to vibration functions[6] or accelerometers. In this section, we describe how we achieved communication between phones and between phones and the server, what library and parameters we used to perform cryptographic operations, and how users run the protocol.

Communication between phones is implemented with the `android.os.Vibrator` class to vibrate a phone, the `android.hardware.SensorManager` to access the phone's accelerometer, and an `android.hardware.SensorEventListener` to learn when the accelerometer has new data. With the DROID, we used a simple on/off keying and tested varying bit lengths from 60 ms per bit to 100 ms per bit to test the reliability of the channel (see Section 8.2). For example, with 100 ms/bit, the phone vibrates for 100 ms to transmit a 1. We use Reed-Solomon encoding to allow the Verifier to recover from errors during reception of the pair identifier. We use 8-bit symbols and include 4 error-correction symbols. This allows the Verifier to recover from errors in 2 independent bytes. When sending the confirmation, the Verifier transmits 2 bytes of 1s to indicate "yes" or sends 0xC003 (2 ones, 12 zeros, 2 ones) to indicate "no". Given the redundancy in the message, the

[5]Audio engineers suggest using accelerometers as "contact microphones" to reduce pickup from sources that are not in direct physical contact [19].

[6]For example, Apple limits the vibration functionaility accessible for applications on the App Store.

Verifier simply transmits the two bytes without any error-correction. The Endorser only considers the confirmation a "yes" if the confirmation contains a sufficiently long series of consecutive 1s.

Communication with the server uses TCP sockets. The phone connects to the server at the beginning of the protocol and maintains a single TCP connection.

All of the cryptographic operations use the Bouncy Castle[7] Java package. We use SHA-1 to create the pre-authenticator and to verify the public key. 1024-bit RSA signatures are used to sign and verify the exchanged information. One could generate a new RSA key pair for each exchange. Instead, we generate a key pair during the first execution and save the key pair for future executions to keep subsequent execution times shorter and more consistent.

To run the protocol, all the users have to do is hold the phones together back-to-back with one phone facing up and press a button to start the exchange. This orientation provides good transmission between the phones' vibrators and accelerometers and allows the phones to automatically assign Endorser or Verifier roles. Currently, the phone with the screen facing down becomes the Verifier. The users hold the phones together until the phones beep to indicate completion. The phones play a lighter tone to indicate success and a harsher sound to indicate a failed exchange. These tones are selected such that they do not alter the confirmation on the vibration channel.

7.2 Java-Based Server

Instead of using local communication (Bluetooth or ad hoc WiFi), Shot uses a server on the Internet to transfer the majority of data between phones. Our server is a Java program running on a desktop machine that listens for incoming connections and uses a database to store and look up information based on the received hash value.

The server does not verify if any of the information is correct (e.g., hash a potential endorser key to verify it matches the pair ID). The phones perform all of the verification.

8. EVALUATION

In this section, we evaluate the reliability of the vibration-to-accelerometer channel used in Shot, and compare the performance of Bump on both iPhone and DROID to Shot on DROID. We also present some microbenchmarks to help explain differences in the execution times.

8.1 Microbenchmarks

Table 1 shows the average time required to perform network and cryptographic operations on an iPhone 3G with iOS 4.2.1 and a DROID with Android 2.2. Each value represents the average from 20 executions of the operation. We measured the time needed to connect to the Bump server over WiFi and send a kilobyte of information using TCP without TLS, and the time needed to sign a random value and verify the signature with 1024-bit RSA keys. The phones exhibit similar network capabilities and require roughly 90 ms to establish a connection and send the data. However, the DROID appears to have greater processing power. Signing is almost five times faster on the DROID than on the iPhone (41.9 ms versus 201.5 ms). Verifying is also faster on the DROID (7 ms versus 8.4 ms).

[7]`http://www.bouncycastle.org/`

Platform	*Connect & Send 1kB*	*1024 RSA Sign*	*1024 RSA Verify*
iPhone	90.0 ms	201.5 ms	8.4 ms
DROID	91.2 ms	41.9 ms	7.0 ms

Table 1: Network and cryptographic performance.

Milliseconds per bit	60	80	100
Rate of successful transmissions	0.53	0.67	0.94

Table 2: Impact of data rate on transmission reliability.

8.2 Reliability of the Vibration Channel

To measure the reliability of the vibration channel, we attempted to exchange the 112-bit message consisting of a pre-authenticator and error-correcting codes while varying the time needed to communicate one bit from the vibrating phone to the other phone's accelerometer from 60 to 100 ms and recorded how often the receiving phone could successfully decode the message. The results of this test are presented in Table 2. At 60 ms/bit, the receiver was able to successfully decode the message 53% of the time. At 100 ms/bit, decoding was successful all but one time. The limiting factor for using vibration and accelerometers for communication appears to be the scheduling on the DROID: the vibrator would not turn on and off promptly for intervals smaller than 75 ms. Future smartphones with more control over the vibrator and more sensitive accelerometers will enable more reliable delivery with smaller intervals. However, more accurate accelerometers may increase the accuracy of smartphone-based attacks that monitor vibrations to detect keystrokes on the phone [4] or nearby keyboards [16].

8.3 Complete Protocol Execution Times

We measured the execution time of Bump (v2.4.0 on iPhone and v1.3.2 on Droid) and Shot on Droid over the course of 10 successful exchanges. During our evaluation, all of the wireless communication was sent over the WiFi network. For Bump, we measured execution time from when the application was started until the phone received the other phone's information. For Shot, we measured the execution time from when the application was started to the tone at the end of Shot. Figure 5 summarizes the results of our evaluation.

The performance of Bump is highly platform dependent, requiring an average of 21.0 seconds on the DROID and 10.4 seconds on the iPhone. The DROID appears to have faster processing and similar network performance (see Section 8.1), but appears to take longer to determine its location. Our experiments were performed indoors and the iPhone appears to quickly switch from trying to use GPS to using WiFi to estimate its location before connection to the Bump server. However, the DROID version of Bump appeared to spend a variable amount of time trying to use GPS before switching to using WiFi.

With fixed-size pre-authenticators and confirmation vibration, Shot provides different execution times depending on the encoding used. When compared to Bump on the iPhone, Shot provides similar execution times when using the the less reliable, 60 ms/bit encoding (9.9 to 12.8 seconds). However, the more reliable version of Shot with 100 ms/bit encoding (15.1 to 16.9 seconds) is slower than the

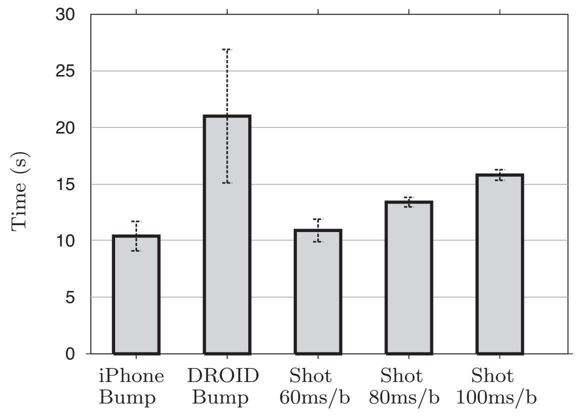


Figure 5: Average execution time for Bump and Shot (error bar = one standard deviation).

version of Bump for the iPhone, but still faster than the DROID version Note that our implementation of Shot was not optimized for efficiency, and the purpose of this evaluation was simply to confirm that Shot could be executed roughly as quickly as Bump.

9. RELATED WORK

Several works have examined the problem of how to exchange information or establish a shared secret between two devices to secure the exchange of information. We discuss different categories of previous work according to the mechanism used to provide security and discuss the user-involvement and security of each.

Human Assisted Comparison A number of prior works require the user to perform a comparison after the exchange to detect an attack [2, 7, 12, 21, 26]. Since an attacker is unable to change the output on the screens or the value the user enters into a device, this technique can successfully detect MitM attacks. Many works focus on comparing a string of hexadecimal digits [2, 12, 26]. However, Uzun et al. [25] found usability issues with string comparisons (i.e., users failed to detect attacks or indicated the strings were different when they were the same). Other works have proposed encoding the comparison value into a sentence [7] or image [21] to improve usability. However, these schemes still require users to perform a comparison and are vulnerable to attacks when users click "Accept" without comparing the strings, sentences, or images.

Exchange of Secrets A number of works assume the attacker is unable to observe the users talking in the room (e.g., sharing a password [2]), motion of the phones [6, 10, 13, 17], communication over the vibration channel [9, 23], or communication over an electrical connection between the devices [24]. Our threat model includes a more powerful attacker who may be able to violate the secrecy of all but Stajano and Anderson's protocol [24]. Nearby parties can eavesdrop on spoken communication to recover a password. Our attack on Bump demonstrated how attackers can observe and imitate simple movements. Mayrhofer and Gellersen showed that it is possible to tune exchange algorithms to be resistant to this kind of attack, but at significant cost to the usability of the system: their system experienced a 10% failure rate of legitimate exchanges, and 16% of participants were unable to ever successfully complete the protocols [17]. Beyond sacrificing usability to provide security against a simple attacker, these protocols may be vulnerable to more sophisticated attacks, such as those that use real-time motion tracking [15] to reconstruct movements than an unaided human could not. When the phones' vibration function is used, user involvement is limited since users only have to hold the phones together, but it is still possible for an attacker to eavesdrop on the vibration channel to acquire the secret [8]. Stajano and Anderson use a wire or other electrical connection between the two devices [24]. In that scenario, the only way an attack will succeed is if the attacker can control communication on the wire without the users noticing. The drawback to this approach is that users have to carry around cables in order to perform the exchange.

Observable Channel Between Devices Much like Shot, other works have examined the use of a channel between two devices that allow users to infer which parties are communicating. Prior works considered using IR [1] or light in the visual spectrum [18, 22] as authentic human-observable channels. If the hardware exists, these techniques can be used on smartphones. However, IR is not available on Blackberry, iPhone, or Android phones, which account for over 75% of all smartphones.[8] Seeing-is-Believing (SiB) [18] and Saxena's follow-up work [22] use the phone's camera to photograph barcodes or film a blinking light. However, SiB is a directional exchange, so users have to execute the protocol twice. Saxena's protocol allows a complete exchange with a single execution of the protocol. However, in addition to filming on P_A, the user has to confirm the exchange by pressing a button on P_B. Shot allows phones to exchange authentic information in both directions, without involving the users to reposition the phones or press a button, if both phones have vibrators and accelerometers.

10. CONCLUSION

The exchange of information between smartphones allows users to play games, share contact information, and even transfer money. In this paper, we presented a man-in-the-middle (MitM) attack against Bump, the most popular smartphone exchange protocol. We analyzed our attack under real-world settings and found that in some realistic scenarios a malicious party can launch a MitM attack against Bump and succeed 70% of the time.

We also presented Shot, a new secure and simple-to-use smartphone exchange. Shot uses the phones' accelerometers and vibrators to establish a human-observable channel between two phones that are held together. Since users can feel the vibrations, this channel provides demonstrative identification of the devices participating in the exchange, and allows users to detect if an attack is occurring (e.g., a remote party is trying to vibrate the phones). Shot leverages asymmetric cryptography to bootstrap authentic communication over the higher-bandwidth wireless channel.

We implemented Shot and compared its execution time and user experience to Bump. Our evaluation found that Shot is comparably quick to Bump, while providing greater security and placing similar demands on the user. As such,

[8]`http://blog.nielsen.com/nielsenwire/online_mobile/mobile-snapshot-smartphones-now-28-of-u-s-cellphone-market/`

Shot represents a new point on the spectrum of exchange protocols, providing improved security or reduced user involvement when compared to existing solutions.

Acknowledgements

This research was supported in part by CyLab at Carnegie Mellon under grants DAAD19-02-1-0389 and W911NF-09-1-0273 from the Army Research Office; by grant W911NF-07-1-0287 from the Army Research Office; by NSF grant CNS-0831407; and by General Motors through the GM-CMU Collaborative Research Laboratory.

11. REFERENCES

[1] Balfanz, D., Smetters, D., Stewart, P., and Wong, H. C. Talking to strangers: Authentication in ad-hoc wireless networks. In *Proceedings of the Network and Distributed System Security Conference (NDSS)* (2002).

[2] Bluetooth Special Interest Group. Simple Pairing Whitepaper, revision v10r00. `http://www.bluetooth.com/NR/rdonlyres/0A0B3F36-D15F-4470-85A6-F2CCFA26F70F/0/SimplePairing_WP_V10r00.pdf`, August 2006.

[3] bump Technologies. Bump. `http://bu.mp`.

[4] Cai, L., and Chen, H. Touchlogger: Inferring keystrokers on touch sceen from smartphone motion. In *Proceedings of the USENIX Workshop on Hot Topics in Security (HotSec)* (2011).

[5] Carbone, M., and Rizzo, L. Dummynet revisited. *SIGCOMM Computer Communications Review 40* (April 2010).

[6] Castelluccia, C., and Mutaf, P. Shake them up! A movement-based pairing protocol for cpu-constrained devices. In *Proceedings of the ACM International Conference on Mobile Systems, Applications, and Services (MobiSys)* (2005).

[7] Goodrich, M. T., Sirivianos, M., Solis, J., Tsudik, G., and Uzun, E. Loud and clear: Human-verifiable authentication based on audio. In *International Conference on Distributed Computing (ICDCS)* (2006).

[8] Halevi, T., and Saxena, N. On pairing constrained wireless devices based on secrecy of auxiliary channels: The case of acoustic eavesdropping. In *Proceedings of the ACM Conference on Computer and Communications Security (CCS)* (2010).

[9] Heydt-Benjamin, T. S., Ransford, B., Clark, S. S., Defend, B., Morgan, W., Fu, K., Kohno, T., and Maisel, W. H. Pacemakers and implantable cardiac defibrillators: Software radio attacks and zero-power defenses. In *Proceedings of the IEEE Symposium on Security and Privacy* (2008).

[10] Holmquist, L. E., Mattern, F., Schiele, B., Alahuhta, P., Beigl, M., and Gellersen, H.-W. Smart-its friends: A technique for users to easily establish connections between smart artefacts. In *Proceedings of Ubicomp* (2001).

[11] Kumar, A., Saxena, N., Tsudik, G., and Uzun, E. Caveat emptor: A comparative study of secure device pairing methods. In *Proceedings of the IEEE International Conference on Pervasive Computing and Communications (PerCom)* (2009).

[12] Laur, S., and Nyberg, K. Efficient mutual data authentication using manually authenticated strings. In *Cryptology and Network Security (CANS)* (2006).

[13] Lester, J., Hannaford, B., and Gaetano, B. "are you with me?" – Using accelerometers to determine if two devices are carried by the same person. In *Proceedings of Pervasive* (2004).

[14] Lin, Y.-H., Studer, A., Hsiao, H.-C., McCune, J., Wang, K.-H., Krohn, M., Lin, P.-L., Perrig, A., Sun, H.-M., and Yang, B.-Y. SPATE: Small-group PKI-less Authenticated Trust Establishment. In *Proceedings of the ACM International Conference on Mobile Systems, Applications, and Services (MobiSys)* (2009).

[15] Lipton, A., Fujiyoshi, H., and Patil, R. Moving target classification and tracking from real-time video. In *Proceedings of the IEEE Workshop on Applications of Computer Vision* (1998).

[16] Marquadt, P., Verma, A., Carter, H., and Traynor, P. (sp)iphone: Decoding vibrations from nearby keyboards using mobile phone accelerometers. In *Proceedings of the ACM Conference on Computer and Communications Security (CCS)* (2011).

[17] Mayrhofer, R., and Gellersen, H. Shake well before use: Intuitive and secure pairing of mobile devices. *IEEE Transactions on Mobile Computing 8*, 6 (2009).

[18] McCune, J. M., Perrig, A., and Reiter, M. K. Seeing-is-believing: Using camera phones for human-verifiable authentication. In *Proceedings of the IEEE Symposium on Security and Privacy* (2005).

[19] O'Reilly, R., Khenkin, A., and Harney, K. Sonic nirvana: Using MEMS accelerometers as acoustic pickups in musical instruments. *Analog Dialogue 43* (Feb. 2009).

[20] PayPal. Paypal mobile payments. `https://personal.paypal.com/us/cgi-bin/?&cmd=_render-content&content_ID=marketing_us/mobile_payments`.

[21] Perrig, A., and Song, D. Hash visualization: A new technique to improve real-world security. In *International Workshop on Cryptographic Techniques and E-Commerce (CrypTEC)* (1999).

[22] Saxena, N., Ekberg, J.-E., Kostiainen, K., and Asokan, N. Secure device pairing based on a visual channel. In *Proceedings of the IEEE Symposium on Security and Privacy* (2006).

[23] Saxena, N., and Watt, J. Authentication technologies for the blind or visually impaired. In *Proceedings of the USENIX Workshop on Hot Topics in Security (HotSec)* (2009).

[24] Stajano, F., and Anderson, R. J. The resurrecting duckling: Security issues for ad-hoc wireless networks. In *Security Protocols Workshop* (1999).

[25] Uzun, E., Karvonen, K., and Asokan, N. Usability analysis of secure pairing methods. In *Proceedings of Usable Security (USEC)* (2007).

[26] Vaudenay, S. Secure communications over insecure channels based on short authenticated strings. In *Advances in Cryptology (Crypto)* (2005).

Attacks on WebView in the Android System[*]

Tongbo Luo, Hao Hao, Wenliang Du, Yifei Wang, and Heng Yin
Dept. of Electrical Engineering & Computer Science, Syracuse University
Syracuse, New York, USA

ABSTRACT

WebView is an essential component in both Android and iOS platforms, enabling smartphone and tablet apps to embed a simple but powerful browser inside them. To achieve a better interaction between apps and their embedded "browsers", WebView provides a number of APIs, allowing code in apps to invoke and be invoked by the JavaScript code within the web pages, intercept their events, and modify those events. Using these features, apps can become customized "browsers" for their intended web applications. Currently, in the Android market, 86 percent of the top 20 most downloaded apps in 10 diverse categories use WebView.

The design of WebView changes the landscape of the Web, especially from the security perspective. Two essential pieces of the Web's security infrastructure are weakened if WebView and its APIs are used: the Trusted Computing Base (TCB) at the client side, and the sandbox protection implemented by browsers. As results, many attacks can be launched either against apps or by them. The objective of this paper is to present these attacks, analyze their fundamental causes, and discuss potential solutions.

1. INTRODUCTION

Over the past two years, led by Apple and Google, the smartphone and tablet industry has seen tremendous growth. Currently, Apple's iOS and Google's Android platforms take 64 percent of the market share, with Android taking 37 percent and iOS 27 percent [8]. Because of the appealing features of these mobile devices, more and more people now own either a smartphone, a tablet, or both. A recent Nielsen survey showed that nearly one third of US mobile users had smartphones at the end of 2010 [8].

A critical factor that has contributed to the wide-spread adoption of smartphones and tablets is their software applications (simply referred to as *apps* by the industry). These apps provide many innovative applications of mobile devices. There are many apps on the market; combined, iOS and Android have over 500,000 apps, for both smartphones and tablets, and the number is still increasing at a fast rate.

Among these apps, many are web-based. Namely, they get contents from web servers using the standard HTTP protocol, display the web contents, and allow users to interact with the web servers. It seems that they are doing exactly what can already be done by real browsers, but there are significant differences. Browsers are designed to be generic, and their features are independent from web applications. Most web-based apps, on the contrary, are customized for specific web applications. Because they primarily serve their intended web applications, they can implement features that are specific to those applications.

For example, `Facebook Mobile` is developed specifically for `Facebook` to provide an easier and better way—compared to `Facebook`'s web interface—to view `Facebook` content, interact with its servers, and communicate with friends. Because of the richer experience gained from these customized "browsers", most users prefer to use them on mobile devices, instead of the actual browsers. Many popular web applications have their dedicated apps, developed in-house or by third parties.

What enables apps to be customized for specific web applications is a technology called *WebView*, adopted by both Android and iOS (it is called UIWebView in iOS, but for simplicity, we simply use WebView throughout this paper). The WebView technology packages the basic functionalities of browsers—such as page rendering, navigation, JavaScript execution—into a class. Apps requiring these basic browser functionalities can simply include the WebView library and create an instance of WebView class. By doing so, apps essentially embed a basic browser in them, and can thus use it to display web contents or interact with web applications. The use of WebView is pervasive. In the Android Market, 86 percent of the top 20 most downloaded Android apps in each of the 10 categories use WebView.

What truly makes customization possible is the APIs provided by WebView. WebView not only allows apps to display web content, more importantly, through its APIs, it enables apps to interact with the web content. The interaction is two-way: From apps to web pages, apps can invoke JavaScript code within web pages or insert their own JavaScript code into web pages; apps can also monitor and intercept the events occurred within web pages, and respond to them. From web pages to apps, apps can register interfaces to WebView, so JavaScript code in the embedded web pages can invoke these interfaces.

[*]This work was supported by Award No. 1017771 from the US National Science Foundation.

With such a two-way interaction mechanism between apps and web pages, apps become more powerful than the traditional browsers. They can customize their interfaces based on the web contents and the screen size, as well as provide additional features beyond what is provided by the web application, giving users a much richer experience than using the generic browsers. For example, `Facebook mobile` makes it easy to stay connected and share with friends, share status updates from the home screen, chat with friends, look at friends' walls and user information, check in to places to get deals, upload photos, share links, check messages, and watch videos. These features, implemented in Java or Object C, are beyond what `Facebook` can achieve with the traditional web interface, through JavaScript and HTML.

Security situations. The pervasive use of WebView and mobile devices has actually changed the security landscape of the Web. For many years, we were accustomed to browsing the Web from a handful of familiar browsers, such as IE, Firefox, Chrome, Safari, etc, all of which are developed by well-recognized companies, and we trust them. Such a paradigm has been changed on smartphones and tablets: thanks to Android's `WebView` and Apple's `UIWebView`, apps can now become browsers, giving us hundreds of thousands "browsers". Most of them are not developed by well-recognized companies, and their trustworthiness is not guaranteed.

A Browser is a critical component in the Trusted Computing Base (TCB) of the Web: Web applications rely on browsers on the client side to secure their web contents, cookies, JavaScript code, and HTTP requests. The main reason why we use those selected browsers is that we trust that they can serve as a TCB, and that their developers have put a lot of time into security testing. When shifting to those unknown "browsers", the trust is gone, and so is the TCB. We do not know whether these "browsers" are trustworthy, whether they have been through rigorous security testing, or whether the developers even have adequate security expertise. Therefore, WebView has weakened the TCB of the Web infrastructure.

Another important security feature of browsers is sandbox, which contains the behaviors of web pages inside the browsers, preventing them from accessing the system resources or the pages from other origins. Unfortunately, to support better interactions between apps and web pages, WebView allows apps to punch "holes" on the sandbox, creating a lot of opportunities for attacks.

Overview of our work and contribution. Our work is the first systematic study on the security problems of WebView. The objective of this work is to conduct a comprehensive and systematic study of WebView's impact on web security, with a particular focus on identifying its fundamental causes. Through our systematic studies, we classified some existing concerns that have been raised by the community [3, 5–7] and the new attacks that are discovered by us, based on the cause of the vulnerabilities. These attacks reveal a fundamental problem caused by the weakening of the TCB and sandbox in the WebView infrastructure. Attacks are possible if the apps themselves are malicious, or if they are non-malicious but vulnerable. Android applications and web applications may become victim if they use WebView or are loaded into WebView.

Although we have not observed many attacks related to the relatively new WebView technology, it is just a matter of time before we see them on a large scale. Based on the number of applications that use WebView and the number of people who have downloaded those applications, as we will show in the case study section later, we believe that the impact is quite significant. Both Google's Android and Apple's iOS are vulnerable. In this paper, due to page limitation, we only focus on Android, but we have successfully achieved similar attacks on iOS.

Based on our studies, we discuss how we can continue benefiting from WebView, and at the same time reduce the risks. Since this paper primarily focuses on the attacks and the development of the solutions is still on going, we will leave the solution details to our future paper.

2. SHORT TUTORIAL ON WEBVIEW

In this paper, we will only focus on the Android platform. We first give a brief tutorial on Android's WebView component. On the Android platform, WebView is a subclass of `View`, and it is used to display web pages. Using WebView, Android applications can easily embed a powerful browser inside, using it not only to display web contents, but also to interact with web servers. Embedding a browser inside Android application can be easily done using the following example (JavaScript is disabled by default within WebView):

```
WebView webView = new WebView(this);
webView.getSettings().setJavaScriptEnabled(true);
```

Once the WebView is created, Android applications can use its `loadUrl` API to load a web page if given a URL string. The following code load the `Facebook` page into WebView:

```
webView.loadUrl("http://www.facebook.com");
```

What makes WebView exciting is not only because it serves simply as an embedded browser, but also because it enables Android applications to interact with web pages and web applications, making web applications and Android applications tightly integrated. There are three types of interactions that are widely used by Android applications; we will discuss them in the rest of this section.

2.1 Event monitoring

Android applications can monitor the events occurred within WebView. This is done through the hooks provided by the `WebViewClient` class. `WebViewClient` provides a list of hook functions, which are triggered when their intended events have occurred inside WebView. Once triggered, these hook functions can access the event information, and may change the consequence of the events.

To use these hooks, Android applications should first create a `WebViewClient` object, and then tell WebView to invoke the hooks in this object when the intended events have occurred inside WebView. `WebViewClient` has already implemented the default behaviors—basically doing nothing—for all the hooks. If we want to change that, we can override the hook functions with our own implementation. Let us see the code in the following:

```
WebViewclient wvclient = New  WebViewClient() {
  // override the "shouldOverrideUrlLoading" hook.
  public boolean shouldOverrideUrlLoading(WebView view,String url){
    if(!url.startsWith("http://www.facebook.com")){
      Intent i = new Intent("android,intent.action.VIEW",
                            Uri.parse(url));
      startActivity(i);
    }
  }
  // override the "onPageFinished" hook.
  public void onPageFinished(WebView view, String url) { ...}
}
          webView.setWebViewClient(wvclient);
```

In the example above, we override the `shouldOverrideUrlLoading` hook, which is triggered by the navigation event, i.e., the user tries to navigate to another URL. The modified hook ensures that the target URL is still from `Facebook`; if not, the WebView will not load it; instead, the system's default browser will be invoked to load the URL. In the same example, we have also overridden the `onPageFinished` hook, so we can do something when a page has finished loading.

2.2 Invoke Java from Javascript

WebView provides a mechanism for the JavaScript code inside it to invoke Android apps' Java code. The API used for this purpose is called `addJavascriptInterface`. Android applications can register Java objects to WebView through this API, and all the public methods in these Java objects can be invoked by the JavaScript code from inside WebView.

In the following example, two Java objects are registered: `FileUtils` and `ContactManager`. Their public methods are also shown in the example. `FileUtils` allows the JavaScript code inside WebView to access the Android's file system, and `ContactManager` allows the JavaScript code to access the user's contact list.

```
wv.addJavascriptInterface(new FileUtils(), "FUtil");
wv.addJavascriptInterface(new ContactManager(), "GC");
...
// The FileUtils class has the following methods:
public int write (String filename, String data, boolean append);
public String read (filename);
...
// The ContactManager class has the following methods:
public void searchPeople (String name, String number);
public ContactTriplet getContactData (String id);
...
```

Let us look at the `FileUtils` interface, which is binded to WebView in the name of `FUtil`. JavaScript within the WebView can use `FUtil` to invoke the methods in `FileUtils`. For example, the following JavaScript code in a web page writes its data to a local file through `FUtil`.

```
<script>
  filename = '/data/data/com.livingsocial.www/' + id +'_cache.txt';
  FUtil.write(filename, data, false);
</script>
```

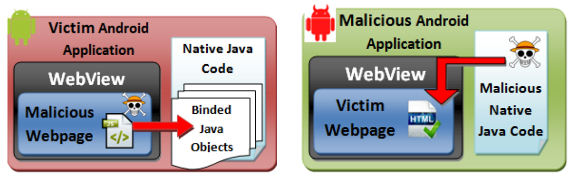
2.3 Invoke JavaScript From Java

In addition to the JavaScript-to-Java interaction, WebView also supports the interaction in the opposite direction, from Java to JavaScript. This is achieved via another WebView API called `loadUrl`. If the URL string starts with `"javascript:"`, followed by JavaScript code, the API will execute this code within the context of the web page inside WebView. For example, the following Java code adds a "Hello World" string to the page, and then sets the cookie of the page to empty.

```
String str="<div><h2>Hello World</h2></div>";
webView.loadUrl("javascript:document.appendChild(str);");
webView.loadUrl("javascript:document.cookie='';");
```

It can be seen from the above example that the JavaScript code has the same privileges as that in the web page: they can manipulate the page's DOM objects and cookies, invoke the JavaScript code within the page, send AJAX requests to the server, etc. Using `loadUrl`, Android applications can extend the functionalities of web applications, giving users a much richer browsing experience.

3. THREAT MODELS



(a) Malicious Web Pages (b) Malicious Apps

Figure 1: Threat Models

The attacks discussed in this paper are categorized based on two threat models, depicted in Figure 1. We give a high-level overview of these models in this section, leaving the attack details to later sections. It should be noted that we will not discuss the attacks that are common in the Web, such as cross-site scripting, cross-site request forgery, SQL injection, etc., because these attacks are not specific to WebView: WebView is not immune to them, nor does it make the situation worse.

Attacks from Malicious Web Pages. We study how malicious web pages can attack Android applications. In this attack model, we assume that apps are benign, and they are intended to serve a web application, such as `Facebook`. These apps can be both first-party (owned by the intended web application) and third-party (owned by an independent entity). The objective of attackers is to compromise the apps and their intended web application. To achieve this, the attackers need to trick the victim to load their web pages into the apps, and then launch attacks on the target WebView. The attack is depicted in Figure 1(a). Getting the victim to load attacker's web pages is not very difficult, and it can be done through various means, such as emails, social networks, advertisements, etc.

Attacks from Malicious Apps. We study how malicious apps can attack web applications. In this threat model, we assume that an attacker owns a malicious app, designed specifically for a web application, e.g., `Facebook`. The goal of the attacker is to directly launch attacks on the web application. The attack is depicted in Figure 1(b). Obviously, these attacks only make sense for third-party apps. To prepare for such attacks, the attacker needs to allure users to use their apps for the intended web application.

Although sounded difficult, the above goal is not difficult to achieve at all, and many apps from the Android market have already achieved that, although none of them is malicious to the best of our knowledge. For example, one of the most popular `Facebook` apps for Android is called `FriendCaster for Facebook`, which is developed by `Handmark`, not `Facebook`; it has been downloaded for `500,000` times. The app uses WebView to browse `Facebook`.

4. ATTACKS FROM WEB PAGES

4.1 Attacks through Holes on the Sandbox

Among all WebView's APIs, addJavascriptInterface is probably the most interesting one. It enables web application's JavaScript code to invoke Android application's Java code (or iOS application's Objective-C code). Section 2 has already given examples on how the API is used.

Allowing apps to bind an interface to WebView fundamentally changes the security of browsers, in particular, it breaks the sandbox model adopted by all browsers. Because of the risk of running untrusted JavaScript programs inside browsers, all browsers implement an access control mechanism called *sandbox* to contain the behaviors of these programs. The sandbox basically achieves two objectives: isolate web pages from the system and isolate the web pages of one origin from those of another. The second objective mainly enforces the Same-Origin Policy (SOP).

When an application uses addJavascriptInterface to attach an interface to WebView, it breaks browser's sandbox isolation, essentially creating holes on the sandboxes. Through these holes, JavaScript programs are allowed to access system resources, such as files, databases, camera, contact, locations, etc. Once an interface is registered to WebView through addJavascriptInterface, it becomes global: all pages loaded in the WebView can call this interface, and access the same data maintained by the interface. This makes it possible for web pages from one origin to affect those from others, defeating SOP.

Opening holes on the sandbox to support new features is not uncommon. For example, in the previous Web standard, the contents in two frames with different domains are completely isolated. Introducing cross-frame communication for mashup applications to exchange data opens a hole on the sandbox. However, with the proper access control enforced on the hole, this new feature was perserved and protected. The WebView's new feature, however, was not properly designed. The objective of this paper is not against this feature, on the contrary, by pointing out where the fundamental flaw is, we can preserve Web's feature and at the same time make it secure.

Attacks on the System. We will use DroidGap [2] as an example to illustrate the attack. DroidGap is not an application by itself; it is an open-source package used by many Android applications. Its goal is to enable developers to write Android apps using mostly WebView and JavaScript code, instead of using Java code. Obviously, to achieve this goal, there should be a way to allow the JavaScript code to access system resources, such as camera, GPS, file systems, etc; otherwise, the functionalities of these apps will be quite limited.

DroidGap breaks the sandbox barrier between JavaScript code and the system through its Java classes, each providing interfaces to access a particular type of system resources. The instances of these Java classes are registered to WebView through the addJavascriptInterface API, so JavaScript code in WebView can invoke their methods to access system resources, as long as the app itself is granted the necessary permissions. The following code shows how DroidGap registers its interfaces to WebView.

```
private void bindBrowser(WebView wv){
   wv.addJavascriptInterface(new CameraLauncher(wv, this), "GapCam");
   wv.addJavascriptInterface(new GeoBroker(wv, this), "Geo");
   wv.addJavascriptInterface(new FileUtils(wv), "FileUtil");
   wv.addJavascriptInterface(new Storage(wv), "droidStorage");  }
```

In the code above, DroidGap registers several Java objects for JavaScript to access system resources, including camera, contact, GPS, file system, and database. Other than the file system and database, accesses to the other system resources need special privileges that must be assigned to an Android app when it is installed. For instance, to access the camera, the app needs to have **android.permission.CAMERA**. Once an app is given a particular system permission, all the web pages—intended or not—loaded into its WebView can use that permission to access system resources, via the interfaces provided by DroidGap. If the pages are malicious, that becomes attacks.

Assume there is an Android app written for Facebook; let us call it MyFBApp. This app uses DroidGap and is given the permission to access the contact list on the device. From the DroidGap code, we can see that DroidGap binds a Java object called ContactManager to WebView, allowing JavaScript code to use its multiple interfaces, such as getContactsAndSendBack, to access the user's contact list on the Android device.

As many Android apps designed to serve a dedicated web application, MyFBApp is designed to serve Facebook only. Therefore, if the web pages inside WebView only come from Facebook, the risk is not very high, given that the web site is reasonably trustworthy. The question is whether the app can guarantee that all web pages inside WebView come from Facebook. This is not easy to achieve. There are many ways for the app's WebView to load web pages from a third party. In a typical approach, the attacker can send a URL to their targeted user in Facebook. If the user clicks on the URL, the attacker's page can be loaded into WebView[1], and its JavaScript code can access the ContactManager interface to steal the user's personal contact information.

Another attack method is through iframes. Many web pages nowadays contain iframes. For example, web advertisements are often displayed in iframes. In Android, the interfaces binded to WebView can be accessed by all the pages inside it, including iframes. Therefore, any advertisement placed in Facebook's web page can now access the user's contact list. Not many people trust advertisement networks with their personal information.

It should be noted that DroidGap is just an example that uses the addJavascriptInterface API to punch "holes" on the WebView's sandbox. As we will show in our case studies, 30% Android apps use addJavascriptInterface. How severe the problems of those apps are depends on the types of interfaces they provide and the permissions assigned to them.

The LivingSocial app is designed for the LivingSocial.com web site. It uses DroidGap, but since the app does not have the permission to access the contact list, even if a malicious page is able to invoke the ContactManager interface, its access to the contact list will be denied by the system. The app is indeed given the permission to access the location information though, so a malicious page can get the user's location using DroidGap's GeoBroker interface.

Attacks on Web Applications. Using the sandbox-breaking addJavascriptInterface API, web applications can store their data on the device as files or databases, something that is impossible for the traditional browsers. Using DroidGap, the LivingSocial app binds a file utility object

[1]There are mechanisms to prevent this, but the app developers have to specifically build that into the app logic.

(FileUtils) to WebView, so JavaScript code in WebView can create, read/write, and delete files—only those belonging to the app—on the device. The LivingSocial app uses this utility to cache user's data on the device, so even if the device is offline, its users can still browse LivingSocial's cached information.

Unfortunately, if the LivingSocial app happens to load a malicious web page in its WebView, or include such a page in its iframe, attackers can use FileUtils to manipulate the user's cached data, including reading, deletion, addition, and modification, all of which are supported by the interfaces provided by FileUtils. As results, the integrity and privacy of user's data for the LivingSocial web application is compromised.

Like LivingSocial, many Android apps use the registered interfaces to pull web application-specific data out of WebView, so they not only cache the data, but also use Java's powerful graphic interface to display the data in a nicer style, providing a richer experience than that by the web interface. The danger of such a usage of addJavascriptInterface is that once the data are out of WebView, they are not protected by the sandbox's same-origin policy, and any page inside, regardless of where it comes from, can access and potentially modify those data through the registered interfaces, essentially defeating the purpose of the same-origin policy.

4.2 Attacks through Frame Confusion

In the Android system, interactions with several components of the system are asynchronous, and require a callback mechanism to let the initiator know when the task has completed. Therefore, when the JavaScript code inside WebView initiates such interactions through the interface binded to WebView, JavaScript code does not wait for the results; instead, when the results are ready, the Java code outside WebView will invoke a JavaScript function, passing the results to the web page.

Let us use DroidGap's ContactManager interface as an example: after the binded Java object has gathered all the necessary contact information from the mobile device, it calls processResults, which invokes the JavaScript function contacts.droidFoundContact, passing the contact information to the web page. The invocation of the JavaScript function is done through WebView's loadUrl API. The code is shown in the following:

```
public void processResults(Cursor paramCursor){
 string result = paramCursor.decode();
 string str8 = new StringBuilder().append("javascript:
              navigator.contacts.droidFoundContact(...)").
 localWebView.loadUrl(str8);
}
```

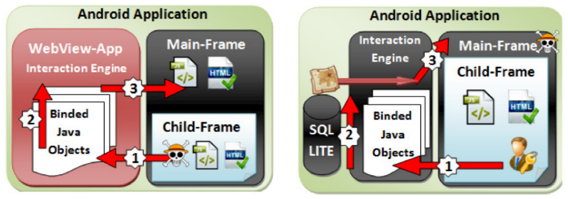
The JavaScript function contacts.droidFoundContact in the example is more like a callback function handler registered by the LivingSocial web page. The use of the asynchronous mode is quite common among Android applications. Unfortunately, if a page has frames (e.g. iframes), the frame making the invocation may not be the one receiving the callback. This interesting and unexpected property of WebView becomes a source of attacks.

Frame Confusion. In a web page with multiple frames, we refer to the main web page as the main frame, and its embedded frames as child frames. The following example demonstrates that when a child frame invokes the Java interface binded to the WebView, the code loaded by loadUrl is executed in the context of the main frame.

```
Object obj = new Object(){
  public void showDomain()
    {mWebView.loadUrl("javascript:alert(document.domain)");}
};
mWebView.addJavascriptInterface(obj, "demo");
```

The code above registers a Java object to the WebView as an interface named "demo", and within the object, a method "showDomain" is defined. Using loadUrl, this method immediately calls back to JavaScript to display the domain name of the page.

When we invoke window.demo.showDomain() from a child frame, the pop-up window actually displays the domain name of the main frame, not the child frame, indicating that the JavaScript code specified in loadUrl is actually executed in the context of the main frame. Whether this is an intended feature of WebView or an oversight is not clear. As results, the combination of the addJavascriptInterface and loadUrl APIs creates a channel between child frames and the main frame, and this channel opens a dangerous Pandora's box: if application developers are careless, the channel can become a source of vulnerability, one that does not exist in the real browsers.



(a) Attack from child frame (b) Attack from main frame

Figure 2: Threat Models

Attack from Child Frame. In this attack, we look at how a malicious web page in a child frame can attack the main frame. We use the LivingSocial app as an example. This app loads LivingSocial's web pages into its WebView (in the main frame), and we assume that one of their iframes has loaded the attacker's malicious page. This is not uncommon because that is exactly how most advertisements are embedded. The main objective of the attacker is to inject code into the main frame to compromise the integrity of LivingSocial. Web browsers enforce the Same Origin policy (SOP) by completely isolating the content of the main frame and the child frame if they come from different origins. For example, the Javascript code in the child frame (www.advertisment.com) cannot access the DOM tree or cookies of the main frame (www.facebook.com). Therefore, even if the content inside iframe is malicious, it cannot and should not be able to compromise the page in the main frame.

As we have shown earlier, LivingSocial binds CameraLauncher to its WebView. In this class, a method called failPicture is intended for the Java code to send an error message to the web page if the camera fails to operate.

```
public class CameraLauncher{
  public void failPicture(String paramString){
    String str = "javascript:navigator.camera.fail('";
    str += paramString + "');";
    this.mAppView.loadUrl(str);
  }
}
```

Unfortunately, since `failPicture()` is a public method in `CameraLauncher`, which is already binded to WebView, the method is accessible to the JavaScript code within WebView, from both child and main frames. In other words, JavaScript code in a child frame can use this interface to display an error message in the main frame, opening a channel between the child frame and the main frame. At the first look, this channel may not seem to be a problem, but those who are familiar with the SQL injection attack should have no problem inserting some malicious JavaScript code in 'paramString', like the following:

```
        x'); malicious JavaScript code; //
```

As results, the malicious code embedded in `paramString` will now be executed in the main frame; it can manipulate the DOM objects of the main frame, access its cookies, and even worse, send malicious AJAX requests to the web server. This is exactly like the classical cross-site scripting attack, except that in this case, the code is injected through WebView, as illustrated in Figure 2(a).

Attack from Main Frame. In this attack, we look at how a malicious web page in the main frame can attack the pages in its child frames. We still use the `LivingSocial` as an example. We assume that the attacker has successfully tricked the `LivingSocial` app to load his/her malicious page into the main frame of its WebView. Within the malicious page, `LivingSocial`'s web page is loaded into a child frame. The attacker can make the child frame as large as the main frame, effectively hiding the main frame.

Suppose that `DroidGap` uses tokens to prevent unauthorized JavaScript code from invoking the interfaces registered to WebView: the code invoking the interfaces must provide a valid token; if not, the interfaces will simply do nothing. An example is given in the following:

```
public class Storage{
  public void QueryDatabase(SQLStat query, Token token){
    if(!this.checkToken(token))   return;
    else { /* Do the database query task and return result*/ }
  }
}
```

With the above token mechanism, even if the JavaScript code in the malicious main frame can still access the `QueryDatabase` interface, its invocation cannot lead to an actual database query. However, if the call is initiated by the `LivingSocial` web pages—which have the valid token—from the child frame, the invocation is legitimate, and will lead to a query. Unfortunately, when the query results are returned to the caller by the app, using `loadUrl`, because of the frame confusion problem, the query results are actually passed to the main frame that belongs to the attacker. This creates an information-leak channel. Figure 2(b) illustrates the attack.

5. ATTACK FROM MALICIOUS APPS

For the attacks in this section, we assume that attackers have written an intriguing Android application (e.g. games, social network apps, etc.), and have successfully lured users to visit the targeted web application servers from its WebView component.

5.1 The Problem: Trusted Computing Base

As we all know, security in any system must be built upon a solid Trusted Computing Base (TCB), and web security is no exception. Web applications rely on several TCB components to achieve security; an essential component is browser. If a user uses a browser that is not trustworthy or is compromised, his/her security with the web application can be compromised. That is why we must use trusted browsers, such as IE, Firefox, Chrome, Safari, etc.

WebView in the Android operating system changes the TCB picture for the Web, because WebView is not isolated from Android applications; on the contrary, WebView is designed to enable a closer interaction between Android applications and web pages. Using WebView, Android applications can embed a browser in them, allowing them to display web contents, as well as launch HTTP requests. To support such an interaction, WebView comes with a number of APIs, enabling Android application's Java code to invoke or be invoked by the JavaScript code in the web pages. Moreover, WebView allows Android applications to intercept and manipulate the events initiated by the web pages.

Essentially, WebView-embedding Android applications become the "customized browsers", but these browsers, usually not developed by well-recognized trusted parties but potential malicious apps, cannot serve as a TCB anymore. If a web application interacts with a malicious Android application, it is equivalent to interacting with a malicious browser: all the security mechanism it relies on from the browser is gone. In this section, we will present several concrete attacks.

However, this is differnt from the situation when attackers have compromised the whole browser by controlling the native binary code of the browser. In such a situation, attackers control everything in the browser; Malicious Android applications, however, only override the limited portion of the APIs in WebView, and the rest of WebView can still be protected by the underlying system. It is more like the usage of "iFrame", which is used to let websites embed pages from other domains; the web browser enforces the Same Origin Policy to isolate each other if they come from a different domain. Similar to the WebView situation, a malicious webpage can embed a page from Facebook into one of its iframes, the content of the Facebook page will be rendered and displayed. With the underlying access control mechanism enforced by the trusted native browser code, the Facebook page cannot be compromised by its hosting page. Similarly, if WebView is provided to applications as a blackbox (i.e no APIs), it can still be counted as a TCB component for the Web even if it is embedded into a malicious application, because isolation mechanism provided by WebView is implemented using WebKit, which is trustworthy.

5.2 Attack Methods

There are several ways to launch the attacks on WebView. We classified them in two categories, based on the WebView features that were taken advantaged of. The categories, illustrated in Figure 3, are described in the following:

- **JavaScript Injection:** Using the functionalities provided by WebView, an Android app can directly *inject* its own JavaScript code into any web page loaded within the WebView component. This code, having the same privileges as that from the web server, can

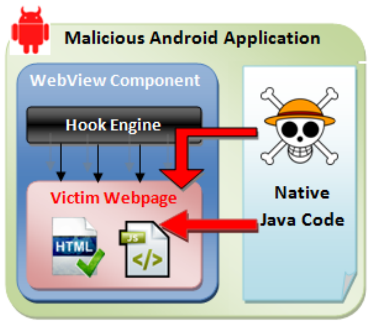


Figure 3: Attack Methods

manipulate everything in the web page, as well as steal its sensitive information.

- **Event Sniffing and Hijacking:** WebView provides a number of hooks (APIs) to Android apps, allowing them to better interact with the web page. Attackers can intercept these APIs, and launch sniffing and hijacking attacks from the outside of WebView, without the needs to inject JavaScript code.

The categories of attacking methods are presented in a decreasing order of severity: if attackers can achieve JavaScript injection, they do not need to use the second method. This indicates that some of the WebView features are more powerful than others. To fully understand the impact of WebView design on security, we study the potential attacks associated with each feature, rather than focusing only on the most powerful feature.

5.3 JavaScript Injection

Using WebView's `loadUrl()` API, Android application can inject arbitrary JavaScript code into the pages loaded by the WebView component. The `loadUrl()` API receives an argument of string type; if the string starts with "javascript:", WebView will treat the entire string as JavaScript code, and execute it in the context of the web page that is currently displayed by the WebView component. This JavaScript code has the same privileges as that included in the web page. Essentially, the injected JavaScript code can manipulate the DOM tree and cookies of the page.

WebView has an option named `javascriptenable`, with `False` being its default value; namely, by default, WebView does not execute any JavaScript code. However, this option can be easily set to `True` by the application, and after that, JavaScript code, embedded in the web page or injected by the application, can be executed.

There are many ways to inject JavaScript code into web page using `loadUrl()`. We give two examples here to illustrate the details.

JavaScript Code Injection. The following Java code constructs a string that contains a short JavaScript program; the program is injected into the web page loaded by WebView. When this program is executed in the context of the web page, it fetches additional (malicious) code from an external web server, and executes it.

```
String js = "javascript: var newscript
     = document.createElement(\"script\");";
js += "newscript.src=\"http://www.attack.com/malicious.js\";";
js += "document.body.appendChild(newscript);";
mWebView.loadUrl(js);
```

In the above example, the malicious code `malicious.js` can launch attacks on the targeted web application from within the web page. For example, if the web page is the user's `Facebook` page, the injected JavaScript code can delete the user's friends, post on his/her friends' walls, modify the user's profiles, etc. Obviously, if the application is developed by `Facebook`, none of these will happen, but some popular `Facebook` apps for Android phones are indeed developed by third parties.

Extracting Information From WebView. In addition to manipulating the contents/cookies of the web page, the malicious application can also ask its injected JavaScript code to send out sensitive information from the page. The following example shows how an Android application extracts the cookie information from a targeted web page [3].

```
class MyJS {
    public void SendSecret(String secret) {
      ... do whatever you want with the secret ...
    }
}
webview.addJavascriptInterface(new MyJS(), "JsShow");
webview.setWebViewClient(new WebViewClient() {
  public void onPageFinished(WebView view, String url){
    view.loadUrl("javascript:
        window.JsShow.SendSecret(document.cookie)");
  }
}
```

In the Java code above, the malicious application defines a class called `MyJS` with a function `SendSecret`, which receives a string as the parameter. The program then registers an instance of `MyJS` to WebView. On finishing loading the page, the application, using `loadUrl`, invokes `window.JsShow.SendSecret`, passing as the parameter whatever sensitive information the attacker wants to extract out of page. In this case, the cookie information is sent out.

5.4 Event Sniffing and Hijacking

Besides the powerful interaction mechanism between Android applications and web pages, WebView also exposes a number of hooks to Android applications, allowing them to intercept events, and potentially change the consequences of events. The `WebViewClient` class defines 14 interfaces [26], using which applications can register event handlers to WebView. When an event was triggered by users inside WebView, the corresponding handler will be invoked; two things can then be done by this handler: observing the event and changing the event, both of which can be achieved from outside of WebView without the need for JavaScript injection.

Event Sniffing: With those 14 hooks, host applications can know almost everything that a user does within WebView, as long as they register an event handler. For example, the `onLoadResource` hook is triggered whenever the page inside WebView tries to load a resource, such as image, video, flash contents, and imported css/JavaScript files. If the host application registers an event handler to this hook, it can observe what resources the page is trying to fetch, leading to information leak. Hooks for other similar web events are described in the following:

- `doUpdateVisitedHistory`: Notify the host Android application to update its visited links database. This hook will be called every time a web page is loaded. Using this hook, Android applications can get the list of URLs that users have visited.

- `onFormResubmission`: Ask the host Android application if the browser should re-send the form. Therefore, the host application can get a copy of the data users have typed in the form.

Using WebView hooks, host applications can also observe all the keystrokes, touches, and clicks that occur within WebView. The hooks used for these purposes include the following: `setOnFocusChangeListener`, `setOnClickListener`, and `setOnTouchListener`,

Event Hijacking: Using those WebView hooks, not only can Android applications observe events, they can also hijack events by modifying their content. Let us look at the page navigation event. Whenever the page within the WebView component attempts to navigate to another URL, the page navigation event occurs. WebView provides a hook called `shouldOverrideUrlLoading`, which allows the host application to intercept the navigation event by registering an event handler to this hook. Once the event handler gets executed, it can also modify the target URL associated with the event, causing the navigation to a different URL. For example, the following code snippet in an Android application can redirect the page navigation to `www.malicious.com`.

```
webview.setWebViewClient(new WebViewClient() {
    public boolean
           shouldOverrideUrlLoading(WebView view, String url)
           { url="http://www.malicious.com";
             view.loadUrl(url); return true;
           }
};
```

The consequence of the above attack is particularly more severe when the victims are trying to navigate to an `"https"` web page, believing that the certificate verification can protect them from redirection attack. This belief is true in the DNS pharming attacks, i.e., even if attacks on DNS can cause the navigation to be redirected to a fraudulent server, the server cannot produce a valid certificate that matches with the URL. This is not true anymore in the above attack, because the URL itself is now modified (not the IP address as in the DNS attacks); the certificate verification will be based on the modified URL, not the original one.

For example, if a page within WebView tries to access `https://www.goodbank.com`, the malicious application can change the URL to `https://www.badbank.com`, basically redirecting the navigation to the latter URL. WebView's certificate verification will only check whether or not the certificate is valid using `www.badbank.com`, not `www.goodbank.com`.

Several other WebView hooks can also lead to the event hijacking attacks. Due to the page limitation and their similarity to the one discussed above, we will not enumerate them in this paper.

6. CASE STUDIES

To understand how risky the situation in Android system is, we turned our attention to the Android Market. Our goal is not to look for malicious or vulnerable apps, but instead to study how Android apps use WebView. We would like to see how ubiquitous the WebView is in Android apps, and how many apps depend on WebView's potentially dangerous features.

6.1 Sample Collection & Methodology

Apps on the Android Market are placed into categories, and we chose 10 in our studies, including Books & Reference, Business, Communication, Entertainment, Finance, News & Magazines, Shopping, Social, Transportation, and Travel & Local. We picked the top 20 most downloaded free apps in each category as the samples for our case studies.

Each Android app consists of several files, all packaged into a single APK file for distribution. The actual programs, written in Java, are included in the APK file in the form of Dalvik bytecode. We use the decompilation tool called `Dex2Jar` [4] to convert the Dalvik bytecode back to the Java source code. Due to the limitations of the tools, only 132 apps were successfully decompiled, and they serve as the basis for our analysis. We realized that `Dex2jar` has some limitations, but it was the best available tool that we could find. Since our case studies are mostly done manually, the limitations of the tool, other than reducing the number of samples, will unlikely affect our results.

6.2 Usage of WebView

We first study how many apps are actually using WebView. We scan the Java code in our 132 samples, looking for places where the WebView class is used. Surprisingly, we have found that 86 percent (113 out of 132) of apps use WebView. We plot our results in Figure 4. Percentage for each category is plotted in Figure 5.

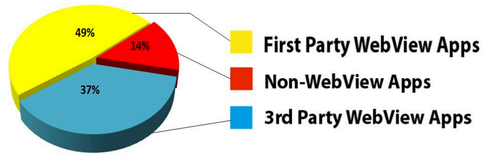


Figure 4: WebView Usage Among Apps

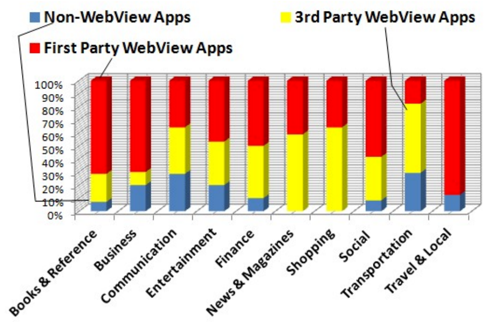


Figure 5: WebView Usage Based On Categories

For the attacks from malicious apps, it only makes sense if the apps and their targeted web applications belong to different entities, i.e., only the third-party apps have motivations to become malicious. Among the 113 apps that use WebView, 49 are third-party apps; despite the fact, these 49 apps are quite popular among users. Based on the data from the Android Market, their average rating is 4.386 out of 5, and their average downloads range from 1,148,700 to 2,813,200. Although these apps are not malicious, they are fully capable of launching attacks on their intended web applications. When that happens, given their popularity, the damage will be substantial.

6.3 Usage of the WebView Hooks

Some of the WebView APIs are security sensitive. To understand how prevalent they have been used, especially by third-party apps, we have gathered statistics on their usage, and depict the results in Figure 6, in which we group them based on the types of attacks we discussed in Section 5.

Among the 49 third-party apps, all use `loadUrl`, 46 use `shouldOverrideUrlloading`, and 25 use `addJavascriptInterface`. We also found that the other APIs, including `doUpdateVisitedHistory`, `onFormResubmission`, and `onLoadResource` are relatively less popular. Overall, our results show that WebView's security-sensitive APIs are widely used. If these apps are malicious, the potential damages are significant.

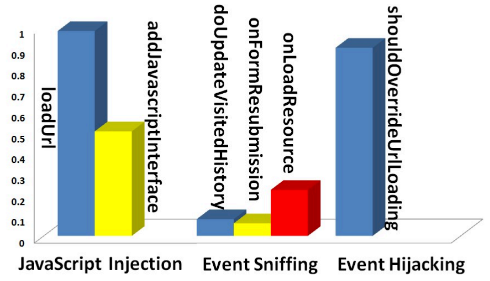


Figure 6: API Usages by Third-Party Apps

6.4 Usage of `addJavascriptInterface`

Attacks from malicious web pages are made possible by the use of the `addJavascriptInterface` API in Android apps, first-party and third-party. We would like to see how many apps actually use this API. We randomly chose 60 apps from our sample pool, decompiled them into Java code, and then searched for the usage of the API. Figure 7 depicts the results, showing that 30 percent of these apps (18 of them) do use the API.

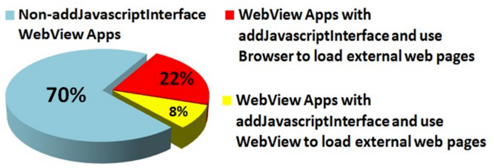


Figure 7: Source Code Investigation

Using the `addJavascriptInterface` API does not automatically make an app potentially vulnerable. To make attacks possible, attackers need to somehow get their malicious pages into the victim's WebView. This goal may not be achievable. WebView provides an hook called `shouldOverrideUrlLoading`, which is triggered every time a navigation event occurs inside WebView. Android apps can implement their own logic to process the navigation event.

Using this hook, apps can restrict what pages can be loaded into WebView, by checking whether the navigation destination URL is allowed or not; if not, they can simply change the URL, or invoke the default browser in the system to display the URL, rather than doing so in WebView. With such a mechanism, an app for `Facebook`, for example, can ensure that all the pages displayed in its WebView are from `Facebook`, essentially preventing malicious external pages from being loaded into WebView.

We have studied the 18 Android apps that use `addJavascriptInterface`, and see how they treat the navigation event. Among them, 7 use the API in the `admob` package, developed by Google for displaying advertisement. Google did a good job in restricting the WebView in `admob` to only display advertisements; if users click on one of the ads, `admob` will invoke the default Android browser to display the target page, not in its WebView. Among the rest 11, which use `addJavascriptInterface` in their own logic, 6 treat the navigation event similarly to `admob`, and the other 5 do allow their WebViews to load external web pages, making them potentially vulnerable. Our results are depicted in Figure 7.

Although using the `shouldOverrideUrlLoading` API does help apps defend against some attacks from malicious pages, it does not work if the malicious pages are inside iframes. The API is only triggered when an navigation event occurs within the main frame of the page, not the child frame. That is, even with the restriction implemented in the API, a page can still load arbitrary external pages within its child frames, making the attacks possible.

7. RELATED WORK

There are several studies focusing on Android's security architecture. The work [9] discussed potential improvement for the Android permission model, empirically analyzed the permissions in 1100 Android applications and visualized them using self-Organizing Map. However, since the attacks we proposed in the paper is not due to the flaw of security model of Android, assigning applications the least of privilege cannot prevent the attacks but only mitigate the impact of the attacks because of the limited privileges granted to the application.

Enck et al. proposes the Kirin security service for Android, which performs lightweight certification of applications to mitigate malware at installation time [12]. Enck et al. also propose "TaintDroid", an efficient, system-wide dynamic taint tracking and analysis system capable of simultaneously tracking multiple sources of sensitive data [11]. Felt et al. have built a tool called "Stowaway", which automatically detects excess privilege when installing third-party Android applications [13]. A systematic analysis of the threats in the Android Market was conducted by [27]. Those studies deal with the fact that mobile systems frequently fail to provide users with adequate control over and visibility into how third-party applications use their private data. The focus of our work is different from these studies: we focus on the security problems of WebView.

With mobile browsers playing more and more important roles in telecommunication [22], browsers themselves have become an active area of research. Microbrowsers designed for surfing the Internet on mobile device become have more and more popular [14]. Initially, research would focus on how to optimize Web content to be effectively rendered on mobile browsers [15, 21]. Recently, a lot of work has focused on analyzing the existing mobile browser models and proposing multiple new models. The paper [28] discusses two patterns of full browsers and C/S framework browsers, and proposes a new collaborative working styles for mobile browsers. The work [24] presents a proxy-based mobile web browser with rich experiences and better visual quality.

Due to the extended use of WebView in Android applications, several Android books [19, 23] have chapters introducing how to use WebView, although none has addressed the security problems of WebView. Some discussions on WebView's security problems can be found at mainstream security-related website like ZDNet [7], and the most relevant discussions were published as blogs [3, 5, 6].

There are numerous studies that focus on enforcing fine-grained access control at the client side, including Caja [1, 10], ConScript [20], Content Security Policy [25], Escudo [16], Contego [17], work by Maffeis at al. [18], etc. Although they were not targeting the problems with WebView, some of their ideas can be extended to defend the attacks on WebView. We will pursue these ideas in our future work.

8. CONCLUSION AND FUTURE WORK

The WebView technology in the Android system enables apps to bring a much richer experience to users, but unfortunately, at the cost of security. In this paper, we have discussed a number of attacks on WebView, either by malicious apps or against non-malicious apps. We have identified two fundamental causes of the attacks: weakening of the TCB and sandbox. Although we have not observed any real attack yet, through our case studies, we have shown that the condition for launching these attacks is already matured, and the potential victims are in the millions; it is just a matter of time before we see real and large-scale attacks.

In our on-going work, we are developing solutions to secure WebView. Our goal is to defend against the attacks on WebView by building desirable security features in WebView.

9. ACKNOWLEDGMENT

We thank the anonymous reviewers and our shepherd, Patrick Traynor, for many detailed and helpful comments.

10. REFERENCES

[1] Caja. http://code.google.com/p/google-caja/.

[2] Droidgap. http://www.phonegap.com.

[3] Extracting html from a webview. http://lexandera.com/2009/01/extracting-html-from-a-webview/.

[4] A tool for converting android's .dex format to java's .class format. http://code.google.com/p/dex2jar.

[5] Injecting javascript into a webview. http://lexandera.com/2009/01/injecting-javascript-into-a-webview/, 2009.

[6] Intercepting page loads in webview. http://lexandera.com/2009/02/intercepting-page-loads-in-webview/, 2009.

[7] Researchers expose android webkit browser exploit. http://www.zdnet.co.uk/news/security-threats/2010/11/08/researchers-expose-android-webkit//-browser-exploit-40090787/, November 2010.

[8] U.S. smartphone market: WhoâĂŹs the most wanted? http://blog.nielsen.com/nielsenwire/, 2011.

[9] David Barrera, H. G üne ş Kayacik, Paul C. van Oorschot, and Anil Somayaji. A methodology for empirical analysis of permission-based security models and its application to android. In *Proceedings of the 17th ACM conference on Computer and communications security*, CCS '10, pages 73–84, New York, NY, USA, 2010. ACM.

[10] D. Crockford. ADSafe. http://www.adsafe.org.

[11] W. Enck, P. Gilbert, B. G. Chun, L. P. Cox, J. Jung, P. McDaniel, and A. N. Sheth. Taintdroid: an information-flow tracking system for realtime privacy monitoring on smartphones. In *Proceedings of the 9th USENIX conference on Operating systems design and implementation*, OSDI'10, pages 1–6, Berkeley, CA, USA, 2010. USENIX Association.

[12] W. Enck, M. Ongtang, and P. McDaniel. On lightweight mobile phone application certification. In *Proceedings of the 16th ACM conference on Computer and communications security*, CCS '09, pages 235–245, New York, NY, USA, 2009. ACM.

[13] A. P. Felt, E. Chin, S. Hanna, D. Song, and D. Wagner. Android permissions demystified, 2011.

[14] E. A. Hernandez. War of the mobile browsers. *IEEE Pervasive Computing*, 8:82–85, January 2009.

[15] A. Jaaksi. Developing mobile browsers in a product line. *IEEE Software*, 19:73–80, 2002.

[16] K. Jayaraman, W. Du, B. Rajagopalan, and S. J. Chapin. Escudo: A fine-grained protection model for web browsers. In *Proceedings of the 30th International Conference on Distributed Computing Systems (ICDCS)*, Genoa, Italy, June 21-25 2010.

[17] T. Luo and W. Du. Contego: Capability-based access control for web browsers. In *TRUST'11*, 2011.

[18] S. Maffeis, J. C. Mitchell, and A. Taly. Object capabilities and isolation of untrusted web applications. In *IEEE Symposium on Security and Privacy*, 2010.

[19] D. McMahon. Learn android programming, 2011.

[20] L. A. Meyerovich and B. Livshits. Conscript: Specifying and enforcing fine-grained security policies for javascript in the browser. In *IEEE Symposium on Security and Privacy*, pages 481–496, 2010.

[21] M. Palviainen and T. Laakko. Mimeframe - a framework for statically and dynamically composed adaptable mobile browsers. 2006.

[22] F. Reynolds. Web 2.0-in your hand. *IEEE Pervasive Computing*, 8:86–88, January 2009.

[23] S. Hashimi S. Komatineni, D. MacLean. Pro android 3, 2011.

[24] H. Shen, Z. Pan, H. Sun, Y. Lu, and S. Li. A proxy-based mobile web browser. In *Proceedings of the international conference on Multimedia*, MM '10, pages 763–766, New York, NY, USA, 2010. ACM.

[25] S. Stamm, B. Sterne, and G. Markham. Reining in the web with content security policy. In *WWW*, 2010.

[26] Android Development Team. Webviewclient hooks list. http://developer.android.com/reference/android/webkit/WebViewClient.html.

[27] T. Vennon and D. Stroop. Threat analysis of the android market, 2010.

[28] P. Ye. Research on mobile browser's model and evaluation. *Structure*, pages 712–715, 2010.

Mitigating Code-Reuse Attacks with Control-Flow Locking

Tyler Bletsch   Xuxian Jiang   Vince Freeh
Department of Computer Science,
NC State University, Raleigh, NC, USA
{tkbletsc, xuxian_jiang, vwfreeh}@ncsu.edu

ABSTRACT

Code-reuse attacks are software exploits in which an attacker directs control flow through existing code with a malicious result. One such technique, return-oriented programming, is based on "gadgets" (short pre-existing sequences of code ending in a `ret` instruction) being executed in arbitrary order as a result of a stack corruption exploit. Many existing code-reuse defenses have relied upon a particular attribute of the attack in question (e.g., the frequency of `ret` instructions in a return-oriented attack), which leads to an incomplete protection, while a smaller number of efforts in protecting *all* exploitable control flow transfers suffer from limited deployability due to high performance overhead. In this paper, we present a novel cost-effective defense technique called *control flow locking*, which allows for effective enforcement of control flow integrity with a small performance overhead. Specifically, instead of immediately determining whether a control flow violation happens before the control flow transfer takes place, control flow locking lazily detects the violation after the transfer. To still restrict attackers' capability, our scheme guarantees that the deviation of the normal control flow graph will only occur *at most* once. Further, our scheme ensures that this deviation cannot be used to craft a malicious system call, which denies any potential gains an attacker might obtain from what is permitted in the threat model. We have developed a proof-of-concept prototype in Linux and our evaluation demonstrates desirable effectiveness and competitive performance overhead with existing techniques. In several benchmarks, our scheme is able to achieve significant gains.

1. INTRODUCTION

Computers are under constant threat of being compromised by increasingly sophisticated attackers. In the context of network daemons, attackers send maliciously crafted packets which exploit software bugs in order to gain unauthorized control. Research into preventing the existence of such bugs has been ongoing, but so far has failed to yield a silver bullet to the problem of exploitable security flaws. However, solutions targeting different aspects of the attack itself have had some success; this has led to an arms race between malicious attackers and security researchers.

One of the earliest attack techniques is the *code injection* attack, in which new machine code is written into the vulnerable program's memory, then a bug is exploited to redirect control flow to this new code. Fortunately, the protection technique known as W$\oplus$X, which ensures that memory is either writable or executable (but not both), largely mitigates this attack [1]. However, attackers have responded by employing *code-reuse* attacks, in which a software flaw is exploited to weave control flow through existing code-base to a malicious end. For example, the *return-into-libc* (RILC) technique is a relatively simple code-reuse attack in which the stack is compromised and control is sent to the beginning of an existing libc function [2]. Often, this is used to call `system()` to launch a process or `mprotect()` to create a writable, executable memory region to bypass W$\oplus$X.

To achieve greater expressiveness, a more sophisticated code-reuse attack known as *return-oriented programming* (ROP) was introduced [3]. In this technique, so-called *gadgets* (small snippets of code ending in `ret`) are weaved together in arbitrary ways to achieve Turing complete computation without code injection. In response to this threat, researchers developed defenses which relied upon particular attributes of this attack, such as the the frequency of `ret` instructions [4, 5] or reliance on the stack [6, 7, 8, 9, 10]. Unfortunately, recent evidence has revealed that code-reuse attacks need not rely on the stack or the `ret` instruction to govern control flow, negating these defenses [11, 12]. With that, it is not desirable to continue the trend of developing piecemeal defenses against each new code-reuse variant. Instead, there is a need to enforce *control flow integrity* (CFI), the property that a program's execution is restricted to the *control flow graph* (CFG) dictated by its source code. The CFI property was formalized by Abadi et al. in 2005; this work included a system capable of enforcing CFI in practice [13]. Unfortunately, this system has not seen significant production deployment possibly due to performance concerns.

In order to address this problem, we present an alternative technique, *control flow locking* (CFL), which achieves protection functionally equivalent to CFI enforcement with small performance overhead (e.g., negligible slowdown for many workloads). The key insight that allows for CFL's improved performance over earlier systems is that the validity check is performed *lazily*, i.e. after the actual control flow transfer has occurred. This leads to a better use of the

CPU's split L1 cache, as code need not be loaded as data (see Section 5.2 for details).

Control flow locking is analogous to mutex locking, except instead of synchronization, the lock is asserted to ensure correctness of the control flow of the application. Specifically, a small snippet of *lock* code is inserted before each indirect control flow transfer (`call` via register, `ret`, etc.). This code asserts the lock by simply changing a certain *lock value* in memory; if the lock was already asserted, a control flow violation is detected and the program is aborted. Otherwise, execution passes through the control flow transfer instruction onto the destination. Each valid destination for that control flow transfer contains the corresponding *unlock* code, which will de-assert the lock if and only if the current lock value is deemed "valid". If an invalid code is found (such as a lock from another point of origin), the process will be aborted. In our system, we further take additional precautions to ensure the lock value is not modified by other code, and that unintended instructions cannot be used (see Section 3). Given this, an attacker can now only subvert the natural control flow of the application at most once before being detected. Further, the instructions for system calls can have lock-verification code prepended to them so that this single control flow violation cannot be used to issue a malicious system call. This means that the only consequence of the attack are potential changes to the program's memory state, which is an ability the attacker is already assumed to have based on the threat model (see Section 3).

We have implemented a proof-of-concept control-flow locking prototype on a commodity x86 system. Though our current prototype mainly supports statically-linked programs, it has been shown to offer performance overhead competitive with existing techniques, achieving significant gains in several benchmarks. We believe control-flow locking represents a step further to address code-reuse attacks with a performance penalty small enough for possible deployment in real-world situations. The rest of this paper is organized as follows: Section 2 discusses related work. Section 3 introduces the overall system design and Section 4 presents the implementation. Section 5 evaluates the system in terms of performance and correctness, and Section 6 discusses its implications as a whole. Section 7 concludes this paper.

2. BACKGROUND AND RELATED WORK

In this section, we first present a brief primer on the x86 architecture which readers familiar with that platform may safely skip. After that, we review recent code-reuse attacks as well as existing defenses.

2.1 Background on the x86

To understand the technique presented in this paper, it will be useful to review the technical details of the 32-bit x86 architecture, where our technique has been implemented.

First, note that the x86 assembly language presented is written in AT&T syntax. Registers are written with a percent sign (e.g. `%eax`), literals with a dollar sign (e.g. `$1`), and memory locations as plain symbols (e.g. `k`). Unlike Intel syntax, here the destination operands appear *last*, so `add %edx,%ecx` indicates $ecx \leftarrow ecx + edx$. Memory dereferencing is indicated either with an asterisk (e.g. `*%eax`) or with parenthesis (e.g. `(%eax)`).

Second, instructions are of variable length and are not aligned. This allows for the existence of *unintended code*: new interpretations of existing code achieved when jumping to a non-instruction boundary. For example, if an `add` instruction contains the bytes `0xff`,`0x10` in an operand, one could jump *into* that instruction at those bytes, and the CPU would now interpret the code as `call *%eax`.

Third, the x86 has a hardware-managed stack supported by two CPU registers: `esp`, the top-of-stack pointer, and `ebp`, the bottom-of-frame pointer. In addition to manual manipulation of stack memory, several x86 CPU instructions implicitly affect the stack. For our purposes, the most important are `call`, which pushes the next instruction's address onto the stack, and `ret`, which pops the stack and jumps to the popped address. These instructions are used to maintain control flow for function call and return, respectively. The `ret` instruction is referred to as an *indirect control flow transfer*, meaning that, based on the content of the stack, control flow can be redirected *anywhere*. This is what makes the `ret` instruction vulnerable to exploitation by the return-into-libc and return-oriented programming techniques—simply altering memory by means of a software exploit can lead to arbitrary control flow. In addition to `ret`, there are indirect variants of `call` and `jmp` instructions: they may take a register or memory reference as an operand, allowing a similarly unrestricted change of control flow.

Therefore, to guard against code-reuse attacks in the general case, it is necessary to not only protect `ret`, but also the indirect forms of `call` and `jmp`.

2.2 Code-Reuse Attacks

The return-into-libc (RILC) technique is one of the earliest forms of code-reuse attack. The simplest variant, which calls a single libc function, was documented as early as 1997 [2]. The technique was subsequently refined in order to allow for the chaining of multiple functions [14].

To achieve greater expressiveness from the attack, Shacham introduced return-oriented programming (ROP) in 2007 [3]. In this technique, the attacker seeks out small snippets of code within the existing codebase, each ending in a `ret`. These gadgets may perform memory load/store operations, arithmetic, logic, system calls, etc. By laying out malicious stack content full of ROP gadget addresses, the attacker can execute an arbitrary number of these gadgets. Further, because gadgets may affect the stack pointer itself, it is possible to construct conditional "branches", meaning that execution of return-oriented gadgets need not be linear. This allows for attacker-controlled Turing complete computation in the context of the vulnerable application. ROP has been demonstrated on a variety of platforms, from x86 [3] to RISC [15, 16, 17] to even embedded environments [18]. A variant of ROP has been adapted to the higher-level constructs in the PHP scripting language, culminating in a data stealing exploit as well as an attack on the PHP exception handler that is able to launch to a traditional machine-code-level code-reuse attack [19].

Return-oriented programming has also been applied to the kernel environment: a ROP-based kernel rootkit has been shown to negate all existing kernel integrity protections [20]. Also, this work demonstrated the feasibility of developing ROP attacks in an automated manner based on a ROP "compiler". Further, it has been shown that the codebase needed to achieve Turing completeness in a ROP attack need not be very large. Even the bootloader of an embedded device may have enough gadgets to implement a return-oriented rootkit, which has negative implications for software attestation techniques which seek to ensure program integrity [21].

In response to this, researchers examined the specific attributes that characterize a ROP attack in order to develop corresponding defenses. ROPDefender [7] rewrites binaries to maintain a shadow stack in order to verify each return address. This builds upon previous work in stack protection, including other shadow stack system [8, 9, 10], as well as canary-based stack protection such as StackGuard [22]. Systems like DROP [4] and DynIMA [5] can detect a ROP-based attack based on the short length of ROP gadgets, which results in a very high frequency of `ret` instructions being encountered. The return-less approach even goes so far as to systematically remove every `ret` instruction so that no gadgets can be found in the resulting binary [6].

In response to these defenses, code-reuse attacks were developed which did not rely on `ret`. Checkoway et al. introduced a technique of finding gadgets which end in sequences equivalent to `ret`, such as `pop X ; jmp X` or the Branch-Load-Exchange (BLX) instruction on ARM [12]. Bletsch et al. went further, establishing *jump-oriented programming* (JOP) as a means to maintain malicious control flow without any `ret`-equivalent instructions through the introduction of a special dispatcher gadget [11]. These attacks indicate the need for general defenses, because ad-hoc defense techniques that target specific attributes of specific attacks are insufficient to address the breadth of code-reuse attack techniques available. The existing work in this vein is discussed in the following subsection.

2.3 Existing Defenses

There are several defense techniques proposed to mitigate code reuse attacks. For example, Address-space layout randomization (ASLR) randomizes a process memory layout, making it difficult for an attacker to figure out what pointers to inject into the attack buffer [23, 24, 25, 26]. Unfortunately, ASLR shares its own limitations [14, 27]. In fact, a so-called *de-randomization attack* has been shown to defeat the popular PaX ASLR system in just 216 seconds [28]. Nevertheless, ASLR still raises the bar for attacks and has been widely adopted.

In order to guarantee protection in the face of code-reuse attacks, it is necessary to restrict indirect control flow transfers so that they can only arrive at certain destinations. One of the first techniques seeking to achieve this kind of protection was *program shepherding*, in which a security policy is applied to all control flow transfers [29]. It is implemented on top of a code interpreter framework with a dynamic optimization system to cache native translations of basic blocks. This approach achieves good protection, but had unacceptably high overhead for some workloads (up to 760% in one case).

Abadi et al. introduced the notion of Control Flow Integrity (CFI), which seeks to ensure that execution only passes through approved paths taken from the software's control flow graph [13]. To achieve this, at each indirect jump/call and return instruction, the target address is checked to see if it follows a valid path in the control flow graph. Unfortunately, this particular implementation of CFI suffered from large overhead, and has not seen wide deployment. However, the core idea of enforcing control flow integrity would effectively mitigate the threat of code-reuse attacks, provided it could be achieved at a reasonable cost. Subsequent work on the notion of CFI has allowed for additional security features, including Data Flow Integrity (DFI) [30] and others [31, 32, 33].

Recently, Onarlioglu et al. have introduced G-Free, another system aimed at addressing the threat of code-reuse in the general case [34]. This system works by systematically editing assembly code so that it does not contain indirect control flow transfers as unintended instructions. It can then secure the intended control flow transfers using data protection techniques similar to prior work (e.g. StackGuard [22], etc.). This technique appears to have much improved performance compared to CFI ($\sim 3\%$). However, it is difficult to discern how it performs with control flow intensive benchmarks, as the only evaluation of performance overhead for application-wide protection was on workloads in which control flow was not on the critical path (i.e. IO-bound or computation-kernel based workloads).

The control flow locking technique presented in this work shares many of the same goals as CFI and G-Free, in that it seeks to protect control flow from diversion by attackers. However, it does so in a very different way from these techniques, allowing greatly reduced overhead compared to CFI as well as G-Free in some cases.

3. DESIGN

In this paper, we assume that the attacker can put a payload into memory and exploit a bug to overwrite some control data (a return address, function pointer, etc.), despite the presence of W$\oplus$X protection. Further, we assume that the altered control data will be used at some point to govern control flow. Currently, this data can be used (or abused) to send control flow literally anywhere in the executable region, including unintended code, function entry points, or various return- or jump-oriented gadgets. This freedom on the part of the attacker is what we will be addressing.

The root problem of code-reuse attacks is their promiscuous use of control flow data in general (return addresses, function pointers, etc.). Therefore, there is a need to restrict this data to only valid targets as dictated by the program's control flow graph. To be specific, the control flow operations that must be protected are: (1) unintended code which happens to implement `ret`, `call`, or `jmp`; (2) `ret` instructions; and (3) indirect `call` and `jmp`.

To address the first category, we turn to existing software-fault isolation (SFI) technique introduced by McCamant et al. [35] and refined by the Google Native Client project [36]. This technique effectively eliminates the danger of unintended code. More specifically, because unintended code arises as a result of variable-sized non-aligned instructions, it can be eliminated by imposing alignment artificially. To achieve this, three changes can be applied to the software at the assembly code level. First, no instruction is permitted to cross an n-byte boundary. For the offending ones, no-ops will be inserted to re-align them. Second, all indirect control flow transfers are restricted to targeting n-byte boundaries. Third, all targets for indirect control flow transfers (e.g. post-call sites, function entry points, etc.) are forced to align to an n-byte boundary. These rules, when taken together, ensure that unintended code cannot be reached[1]. In practice, the value of n is a power of two, and control

[1]See [35] for a formal proof of this property.

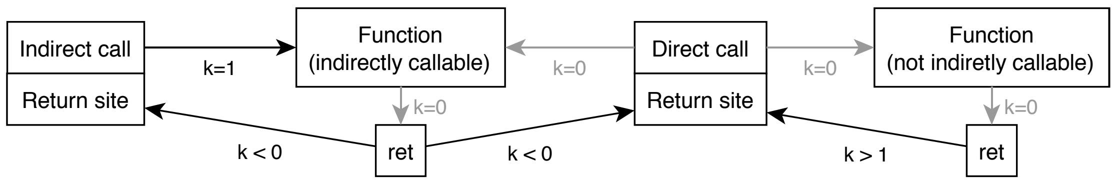


Figure 1: A simplified view of the control flow graph used in the CFL technique. Indirect `jmp` instructions, which generally arise from compiler optimizations, are not shown. The value of the control flow key `k` is shown next to each edge; grey edges do not require control flow locking and are simply present for completeness.

flow transfers are restricted by means of a simple bit mask. The value of n must be larger than any single instruction, and there is a trade-off between smaller n (more frequent instruction alignments) and larger n (longer no-ops before control flow transfers, increased pressure on the CPU instruction cache). In practice, $n = 32$ is a typical value that is chosen when the corresponding transformation is applied at the assembly phase of the build process.

With unintended code removed from consideration, we now focus exclusively on control flow transfers in the rest two categories, i.e., `ret`, indirect `call` and `jmp`. To this end, we apply a novel technique called *control flow locking.*

3.1 Control Flow Locking

To explain this technique, we begin with a degenerate variant of it called *single-bit control flow locking.* In this model, we first locate all indirect control flow transfer sites, which consists of `ret` instructions and the indirect variants of `jmp` and `call`. Before each of these sites, we insert "lock" code, which implements the following pseudo-code:

```
if (k != 0) abort();
k = 1;
```

Here, `k` is known as the *control flow key*, and is simply a word of memory at a fixed location. In this model, the value 0 means "unlocked" and 1 "locked". This is analogous to a mutex lock operation, though atomicity and waiting are not required for our purposes. At each valid indirect control flow target[2] found in the control flow graph, we apply the corresponding "unlock" operation:

```
k = 0;
```

In normal operation, every lock will immediately be followed by a transfer to a corresponding unlock. The attacker's ability to employ a code-reuse exploit, however, has been sharply limited. Control flow from an indirect transfer must pass through an unlock operation before encountering another indirect transfer. Because the only unlock operations correspond to valid transfer targets, this means using coarse-grained pieces of code, such as entire functions. Further, it will not be possible to apply RILC-style whole-function chaining as proposed in [14], because the so-called *esp-lifter* gadget used to connect the functions has been eliminated. Therefore, the only code eligible for use by the attacker are those which are valid indirect transfer targets and happen to themselves include a legitimate control flow transfer.

[2]The exact definition of an indirect jump target varies depending on the code in question. For statically linked code (including the OS kernel) and code destined to be a standalone executable, only those locations explicitly used as function pointers in the source code are considered targets. For dynamic library code, however, any exported function symbol is also a potential entry point.

From the attacker's perspective, this selection is orders of magnitude smaller than the full gamut of gadgets normally available. Moreover, we can make a further addition to make it functionally equivalent to full enforcement of control flow integrity. The above locking algorithms merely ensure that the target of an indirect transfer must pass through *any* valid entry point before jumping again. However, we can extract additional information from the control flow graph to place finer granularity on our enforcement. To this end, `k` can be changed from a single bit to an integer, with values corresponding to paths along the control flow graph. We call this variant *multi-bit control flow locking.* In this model, the lock and unlock algorithms may be rewritten as:

```
lock(value):
   if (k != 0) abort();
   k = value;
unlock(value):
   if (k != value) abort();
   k = 0;
```

Here, `value` is a parameter determined at link-time based on the control flow graph, which limits control flow to valid paths specified in the program's source code. The manner in which the `k` value is selected is covered in the following section.

3.2 Restricted Control Flow Graph

The general meaning of "control flow graph" includes every basic block in the software as nodes and any labels or deviations from linear execution as edges. This includes conditional branches, direct jumps, etc. For the purposes of this work, however, we need only concern ourselves with indirect control flow transfers, i.e. those that jump to a location stored in memory or a register as opposed to in the instruction itself. Therefore, we use a restricted control flow graph which consists of the following classes of nodes: (1) entry points into functions; (2) locations in the code which may be the target for indirect `jmp` or `call`; (3) return sites; and (4) indirect `call` and `ret`.

Accordingly, the graph has the two types of edge: The first type represents the control flow resulting from a `ret` instruction or an indirect `call` or `jmp`. The second type is implied from a function's entry point to each of its `ret` instructions. In Figure 1, we illustrate the two types of edges in our control flow graph. For each black edge, the origin endpoint is the location of "lock" code, and the destination endpoint contains corresponding "unlock" code. The grey edges are direct transfers of control flow, and therefore do not require locking.

Control flow locking is a general mechanism to enforce restrictions on indirect control flow transfers – its accuracy depends on the manner in which `k` is selected at each node and the granularity with which it is matched at each destination. The degenerate *single-bit* variant discussed earlier

k value	Meaning
$\texttt{k} = 0$	Unlocked.
$\texttt{k} = 1$	Indirect `jmp` or `call`.
$\texttt{k} > 1$	Return from a *non-indirectly-callable* function.
$\texttt{k} < 0$	Return from a *indirectly-callable* function.

Table 1: Possible k values in multi-bit CFL.

is equivalent to enforcing $k = 1$ for every black edge. In this work, we evaluate *multi-bit* CFL with a policy derived from static analysis of the source code's control flow graph. The possible values of `k` are enumerated in Table 1.

To protect `ret` instructions in a given function, the lock code before each `ret` sets `k` to a specific value based on which instructions may call it. The specific value is computed as follows. First, we determine the list of direct `call` instructions which refer the function in question. This list is hashed to a produce the value `d`. Second, we determine if the function may be called indirectly. In practice, this means finding if the function's symbol has been used in a data declaration or as an operand to a non-control-flow instruction, such as `mov`. This fact is stored in the boolean `indir`. Based on the above, assuming a word size of 32-bits, the value of `k` is computed as:

```
(indir ? 0x80000000 : 0) | (0x7FFFFFFF & d)
```

The lower 31 bits represent the caller hash `d` with the sign bit representing `indir`. This means that functions which may be called indirectly will have a negative `k` value, while those that cannot will have a positive `k`. (In the unlikely event that `d` = `indir` = 0, a positive `d` will be chosen.) Choosing the values of `k` in this way allows the comparison code to be written in the fewest x86 instructions possible (see Section 4).

In the implementation presented in Section 4, all indirect `call` and `jmp` operations share the `k` value of 1. This is due to a limitation in static code analysis: any symbol that represents a location in code which is also used as data may be stored and referenced in arbitrary ways, through multiple pointers, overlapping data structures, etc. As such, it is not possible to automatically identify which locations an indirect `call` or `jmp` may lead to in general. Therefore, we conservatively assume that any indirect control flow transfer may go to any code location whose symbol is used as data.

This is not a limitation of the CFL technique, however. If additional control flow information can be provided by the programmer or the higher level language (e.g., a more restrictive language than C), then CFL could readily make use of this information to enforce this new finer-grained control flow graph. As originally introduced by Abadi et al., assigning keys to indirect `jmp`/`call` control flow paths can lead to a problem of *destination equivalence* [13]. This occurs because two indirect call sites may have non-disjoint sets of potential destinations. For example, suppose X may call A or B, while Y may call B or C. In this case, list of callers of A, B, and C differ, but B must allow control to flow from either X or Y with a single `k` value. There are three possible solutions to this problem. First, when ambiguity is present, we may assign a single `k` value for all functions involved (e.g., X and Y would share a `k` value). Second, we may apply more fine-grained comparison, such as each destination checking a subset of bits of `k` (e.g., B only checks the lower 16 bits). Third, we may duplicate whole functions, providing different `k` values for the each (e.g., B is replicated as B'). These techniques can be combined depending on the precise structure of the indirect CFG. Due to limitations of static code analysis, however, the implementation presented treats indirect `jmp` and `call` instructions as equivalent. This granularity is sufficient to prevent the jump-oriented programming technique presented in [11], because it precludes the existence of jump-oriented gadgets.

The edges of Figure 1 have been annotated with values for `k`. At each edge origin, the lock code sets `k` to the specified value. At each target, the unlock code verifies that the value of `k` is one of those set by an incoming edge. For example, the return site after a direct call will verify that `k` is equal to the specific key value for the called function, whereas the return site after an indirect call will merely confirm that $\texttt{k} < 0$, as this confirms that control flow was transferred by a `ret` within an indirectly callable function.

Two additional considerations are needed to ensure that control flow locking cannot be bypassed. First, we must ensure that the value of `k` cannot be modified directly through code other than the lock and unlock routines. The x86 architecture has a feature that allows this to be achieved in a straightforward way – memory segmentation. Segmentation permits applications to have multiple separate memory maps which can be uniquely addressed via segment selector registers. Modern operating systems use a flat memory model, and therefore make little use of this feature, meaning that an entire segment register can be dedicated solely to deal with storing `k`. Because there is no unintended code and no explicit segment selection in application-generated code, the only code available to address `k` lies in the lock and unlock routines.

Second, we must ensure that the attacker cannot redirect control flow directly into a system call (e.g., a `sysenter` instruction). This can be achieved simply by prepending lock verification code to each system call instruction which will validate that `k` is in the unlocked state (0). This forces control flow for a system call to pass through the corresponding function entry point.

4. IMPLEMENTATION

To assess the performance impact of the CFL technique, an implementation was developed on a 32-bit x86 Debian Linux 5.0.4 system with an Intel Core 2 Duo E8400 3.0GHz CPU. Because the protection must be applied to complete software stack, a CFL-enabled libc was built based on dietlibc[3] version 0.32 [37]. In addition, gcc itself includes a static library called `libgcc` for inclusion in all binaries – this library contains helper functions unique to each hardware architecture. A CFL-enabled variant of this library was also produced. This implementation is based on statically linked binaries; Section 6 discusses how the technique can be extended to dynamically linked binaries.

4.1 Overview

The CFL system was implemented as two additional phases within the normal gcc build system: (1) an assembly language rewriter and (2) a small post-link patch-up phase. The

[3]This libc variant was designed to minimize code size, but the reason that it was selected for this work is its simplicity and adherence to best practices where assembly code is concerned (such as proper use of locally scoped symbols). These factors simply eased the implementation; there's no reason why a more common libc implementation, such as GNU libc, could not be used instead.

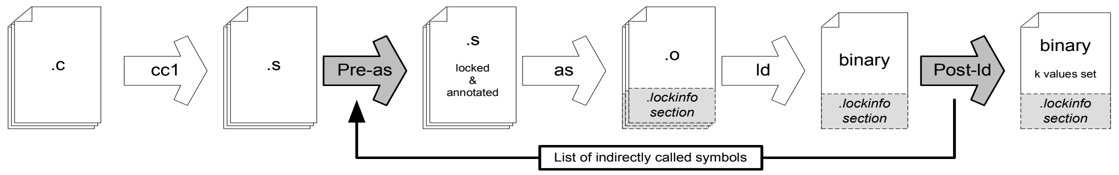


Figure 2: Modified gcc workflow. The grey *Pre-as* and *Post-ld* phases are the only additional steps in the build process.

complete workflow is diagrammed in Figure 2. The assembly language rewriter performs the vast majority of the work, encapsulating both the alignment transformations needed to eliminate unintended code as well as the core CFL transformations. Placing the transformation at the assembly phase allows both C and assembly-language code to be protected, but has the downside of requiring our system to reconstruct some semantic information, such as the call graph, control flow graph, and the list of symbols eligible to be called indirectly. Further, much of this information is not available at level of the individual assembly language files, so it was necessary to implement compilation as a two-pass process.

In the first pass, the assembly rewriter inserts lock and unlock code under the assumption that no symbol may be called indirectly. During this, records of each code symbol, symbol reference, lock operation, and unlock operation are noted in a new ELF section[4] called `.lockinfo`. Then, during the post-link phase, the `.lockinfo` can be used to determine all indirectly callable code symbols. This symbol list is exported for use in the second build pass.

During the second pass, the assembly rewriter can now use the list of indirectly callable code symbols to insert additional unlock operations as needed. As before, all lock and unlock operations are noted in the `.lockinfo` section. The `k` values used in these operations are simply dummy values, as the call graph is not yet known. During the post-link phase of the second pass, the call graph is available, and every `k` value can be computed. The system therefore uses the locations of the lock and unlock operations recorded in the `.lockinfo` to patch in the proper `k` values directly into the x86 code. At this time, the `.lockinfo` section can be discarded to reduce the executable size; it is not needed at runtime.

To summarize, the assembly rewriter phase will: (1) align instructions and function entry points on 32-byte boundaries and restrict control flow instructions to 32-byte boundaries; (2) note all symbol references and code labels in `.lockinfo`; (3) insert lock code before all indirect control flow transfers; and (4) insert unlock code before all indirect control flow destinations (including those found during the post-link phase of the first pass). And the steps undertaken by the post-link patch-up phase are: (1) use the `.lockinfo` to construct the call graph and identify all lock and unlock code locations; (2) make note of indirectly called symbols for use in the second pass; and (3) patch the binary with the `k` values computed for each function in all lock and unlock code.

[4]ELF binary objects in Linux are composed of multiple sections, such a `.text` for code and `.data` for writable data. Arbitrary new sections can be introduced as needed and do not affect normal operation of the program. These sections are present in object files and linked together when building the final binary, at which time any symbols used are resolved.

4.2 Lock/unlock operations

The specific values for `k` (described in Section 3.2) were selected to allow the implementation of lock and unlock operations to be done efficiently. This is achieved by exploiting the difference between signed and unsigned comparisons on the x86. Figure 3 depicts the key assembly code transformations which insert lock and unlock code.

Figure 3(a) shows the lock code inserted before each `ret`. The first two lines ensure that `k` is 0, otherwise aborting with a lock violation error. The third line sets `k` to the proper value for this particular function. At assembly time, this `<key>` is simply set to a dummy value, which is later filled in during the post-link phase.

Figure 3(b) shows the corresponding unlock code for a direct call. This code will ensure that `k` is set to the proper value for the called function before clearing `k` back to 0.

Figure 3(c) shows two transformations. First, because this is an indirect call operation, a lock is inserted before the call. As before, this lock ensures that `k` is 0 (unlocked). It then sets `k` to the proper value for an indirect call (1). After the call returns, a special variant of the unlock code is inserted. This variant must verify that `k` contains a value corresponding to an indirectly callable function, i.e., a negative value. Therefore, the first two lines of this unlock will compare `k` to 0 and abort if $k \geq 0$. Otherwise, `k` will be unlocked.

Figure 3(d) shows the code inserted at the site of a label which may be indirectly called or jumped to. This unlock code corresponds to the lock set in the first top half of Figure 3(c). This code may be reached via an indirect call or jump, in which case $k = 1$, or it may be accessed via a direct call or jump, in which case no locking has taken place and $k = 0$. Therefore, this unlock code must accept only those cases. To achieve this, we switch to using an unsigned compare. To determine if `k` is 0 or 1, we compare it to 1 and abort if and only if $k \overset{u}{>} 1$; otherwise, `k` is unlocked and execution continues.

4.3 Assembly language caveats

It is important to note that, in principle, the concept of a call graph only exists in the realm of higher level languages such as C. Assembly code need not respect this concept. For example, it is possible for one hand-coded (or compiler-optimized) function to jump or fall through into another without using a `call` instruction. In addition, the differentiation between a full-fledged function and a mere label in GNU assembly language is merely based on naming – local labels start with `.L`. However, hand-coders of assembly language need not follow this convention, as the only consequence for making all symbols global is some confusion when using the debugger.

Therefore, it was necessary to ensure that all manually written assembly code (such as that found in dietlibc) be

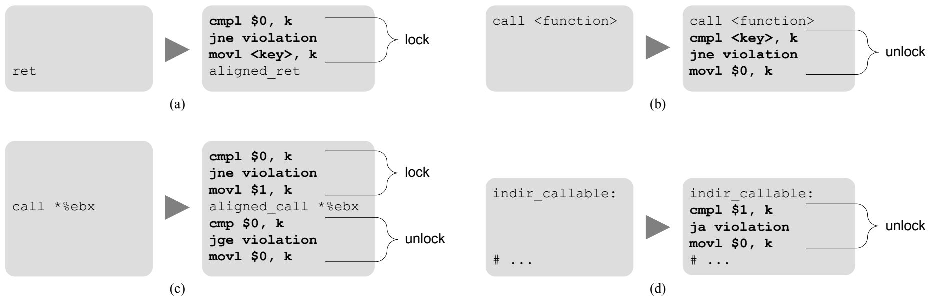


Figure 3: Code transformations responsible for control flow locking. Some transformations for alignment are also shown: the macros `aligned_ret` and `aligned_call` are the same as the normal `ret` and `call` instructions, except they restrict the operand to align to a 32-byte boundary.

made to conform to the same rules as code generated by the C compiler (or at least have the exceptions noted explicitly). To this end, minor changes were made to dietlibc to explicitly indicate fall-through, and direct jumps between functions were detected and automatically annotated as well. When two functions have such a relationship, we say that they are *jump-connected*. Further, jump-connectedness is a transitive relationship, so if A and B are each jump-connected to C, then A is jump-connected to B, and vice versa. Detecting such cases is necessary, because a function may return *on behalf of* another. In this case, the `k` value for the return lock code must match for the two functions.

Therefore, where previously we have made reference to the list of callers of a function, a more accurate description would be the list of callers to all functions jump-connected to the one under consideration.

5. EVALUATION

In this section, we first present the security analysis on the protection offered by CFL. Next, we measure the performance overhead of our prototype.

5.1 Security Analysis

In order to analyze the security provided by CFL, we will review each possible destination for a `ret` instruction or indirect `jmp` or `call` that has been exploited. The possible targets under consideration are the nodes of our indirect CFG (`ret`, indirect `jmp`/`call`, function entry points, and return sites) as well as system call sites. Table 2 enumerates all possible destinations and their eventual outcomes.

In this table, the only outcomes that do not result in the software aborting are those on the valid CFG. The case of control flow arriving "before an indirectly callable function" requires clarification: because functions are n-byte aligned, the only code available *before* a function is the content of the previous function, and all control flow paths in that function must end in a `ret` (or an equivalent operation, such as a direct `jmp` to another function). Therefore, there is no lock-free code path available before a function entry point that would fall through into the function itself.

In Table 2, where "preceding code" may be executed, this cannot include system calls, so the only side effect is a change to program memory and CPU state. However, recall that the attacker's ability to alter CPU and memory state is already assumed, based on the threat model (presented at the start of this section). Therefore, an attacker attempting a code-reuse attack on a CFL-enabled binary achieves no greater control of the program than was provided by the original bug being exploited. In addition, we note that it is possible to mechanically determine if a binary has been properly compiled with CFL protection. This is a straightforward extension of the reliable disassembly method introduced in Native Client [36]. Once the alignment rules have been confirmed, the only extension needed is to verify that all indirect control flow transfers have corresponding lock and unlock code. Because 32-byte alignment eliminates the possibility of unintended code, this check is simply a straightforward scan of the disassembled binary. This means that system-wide CFL protection can be enforced by including such a verification routine in the OS binary loader.

5.2 Performance Analysis

To evaluate the performance impact of CFL, we built a number of C based benchmarks from the SPEC CPU 2000 and SPEC CPU 2006 suites, as well as some common UNIX utilities. The SPEC CPU benchmarks were run using their standard reference workloads. The workload for the UNIX utilities were similar to those used in the evaluation of G-Free [34]: `md5sum` computed the MD5 hash of a 2GB file, `grep` searched a 2GB text file for a short regular expression, and `dd` created a 4GB file on disk by copying blocks from `/dev/zero`. These applications were built using four different assembly rewriter algorithms:

- **None**: No changes made.
- **Just alignment**: Only the alignment rules needed to preclude unintended code are implemented.
- **Single-bit CFL**: The degenerate single-bit jump locking algorithm in which the only values for `k` are 0 and 1. The unlock code is simplified, as it need not check for a locked state, and will instead simple set `k` to 0.
- **Full CFL**: The complete control flow locking scheme.

Overhead was computed for the latter three algorithms compared to "None" as the base case; these results are presented in Figure 4. Overall performance impact can be divided into four categories.

First, many of the workloads (`mcf`, `milc`, `lbm`, `md5sum`, `grep`, and `dd`) exhibited negligible overhead. These applications likely did not perform a large amount of control flow compared to useful computation, i.e., control flow operations were not on the critical path. This may be due to their primary computation being implemented as coarse-grained, long-running functions, or (in the case of `dd`) another resource such as IO being on the critical path.

Second, the compression benchmarks `gzip` and `bzip2` incurred 2–3% overhead just due to alignment, then almost

If control flow is directed...	Then the system will...
at or before a `ret` instruction	run preceding code; abort due to pre-`ret` lock
at or before an indirect `jmp`/`call`	run preceding code; abort due to pre-`call` lock
before an indirectly callable function	abort due to a lock within the previous function
at a valid function entry point (for jmp/call)	proceed normally (valid control flow transfer)
at an invalid indirectly callable function	abort on unlock due to k mismatch
at a valid return site (for ret)	proceed normally (valid control flow transfer)
at an invalid return site	abort on unlock due to k mismatch
before a return site (i.e. before a direct call)	enter function; abort at the next lock operation
at or before a syscall	run preceding code; abort due to pre-syscall trap

Table 2: Possible paths of exploited control flow and their outcomes.

no additional overhead from the inclusion of CFL. That is, the no-ops and address masking operations involved in preventing unintended code accounted for almost the entirety of overhead for these workloads.

Third, the performance of `art` is an interesting case. In single-bit CFL, it incurs almost 5% overhead, but yields near-zero overhead for plain alignment and full CFL. This case is puzzling, and it may be related to an idiosyncrasy of working with the x86 assembly language and microarchitecture. Modern x86 CPUs are super-scalar out-of-order processors with complex branch and value prediction and multiple layers of cache. In addition, some instructions have multiple forms with differing lengths. For example, the conditional jump instruction can be short (encoded in 2 bytes for distances less than 128 bytes) or long (encoded in 6 bytes for larger distances). Therefore, it is possible for assembly-level modifications to have unexpected subtle effects on performance. For example, insertion of a three-instruction "lock" code may make a conditional jump go from short to long form, which in turn may ripple down and affect all subsequent alignment operations, which may in turn alter how the code fits into the CPU instruction cache or the contention for functional units within the ALU. It is very difficult to identify how these subtle changes may affect a given process's execution on a given CPU, so it is not clear that such effects are the necessarily culprit with `art`, but given that full CFL involves strictly *more* inserted code than single-bit CFL, yet has less overhead here, it is the best theory available to explain this case.

Fourth, some workloads (`gap`, `twolf`, and `sjeng`) exhibited significant overhead as more and more instructions were added to govern control flow operations. It is likely that these workloads make use of fine-grained control flow, such as calling many short-lived functions, in the course of their execution. This is supported through application profiling: the `gap` benchmark, which saw the largest CFL overhead, performed over $3.6{\times}10^7$ calls per second, whereas `mcf`, which had negligible overhead, performed only $6.5{\times}10^5$ calls per second. One interesting thing to note is that the CFL technique was applied with no modification to the C optimizer; it may be the case that adjustments could be made to the optimizer to reflect the newly increased cost of control flow operations to mitigate this overhead. For example, the logic that determines when a function should be inlined may need to be re-calibrated to reflect the increased cost of function calls. The question of how to mitigate the performance impact of CFL opens an interesting avenue for future work.

The CFL technique compares favorably against prior techniques. One of the highest overheads recorded for the control flow integrity (CFI) technique proposed by Abadi et al. [13], CPU2000's `gap` benchmark, saw 31% overhead. The CFL technique provides equivalent protection with 21% overhead for this workload. For the other benchmarks available for direct comparison (`bzip2`, `gzip`, and `twolf`), CFI achieved overheads between 0% and 5%; CFL achieves comparable results. A direct comparison between these figures isn't strictly possible, as the CFI work was conducted on a different OS and CPU, and the source code for the system is not available. However, CFL likely compares favorably in the general case because it involves a similar amount of instrumentation, but incurs strictly less L1 data cache pressure compared to CFI. The reason for this is as follows. The x86 L1 cache is split between instructions and data. In CFI, when the destination is checked, memory at the jump target must be loaded as data in the cache. Then, after the comparison succeeds and control moves to the target, the same memory must be loaded again, this time into the instruction side of the L1 cache. In contrast, the CFL technique places the key values as immediate operands within the instructions being executed, removing the need for this double-load behavior. In short, CFL executes similar operations at each control flow transfer while removing needless data cache pressure.

With regard to the G-Free system, making a comparison is difficult because the published evaluation in that work was limited to IO-bound and computation-kernel workloads[5] [34]. As such, without a direct comparison of G-Free across a larger number of workloads, it is difficult to speak generally about the relative performance it versus CFL. Nevertheless, for the metrics that are available, the CFL technique is competitive. Four benchmarks are shared between this work and G-Free's evaluation: `gzip`, `grep`, `dd`, and `md5sum`. For `gzip`, CFL achieved roughly equal overhead to G-Free: 3%. For the others, CFL achieved essentially zero overhead (within the standard deviation), whereas G-Free incurred between 0.6% and 2.6% overhead[6]. These results are promising, but inconclusive. It is possible that neither technique is superior in all cases; there may be a trade-off that makes one of the two schemes more efficient for a given application.

[5]Specifically, the evaluation did end-to-end protection for six workloads: `gzip`, `grep`, `dd`, `md5sum`, `ssh-keygen`, and `lame`. In addition, a wider selection of benchmarks was tested with their technique protecting libc *only*, not the application code. Because control flow is seldom on the critical path within libc, these measurements fail to characterize the performance of the G-Free system with a workload rich in control flow operations, so it is not possible to conclusively compare those workloads to systems like CFL or CFI.

[6]No information on test repetitions or measurement variance was provided in [34].

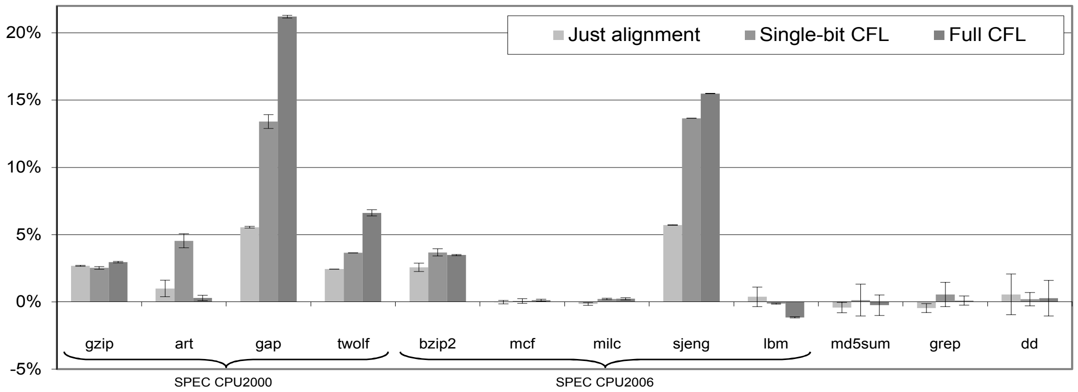


Figure 4: Performance overhead of various forms of the CFL technique.

6. DISCUSSION

It is important to clarify the precise security benefit that the CFL scheme offers: it constrains the path of execution to the software's control flow graph, allowing at most one violation, and guaranteeing that this violation cannot be used to construct a malicious system call directly. In the following, we further elaborate our system and examine possible limitations in our prototype.

First, we highlight that the protection is only as good as the control flow graph being enforced. In this implementation, an automatically derived graph was used. This graph imposed tight restrictions on direct `call`s and their corresponding `ret`s, and limited indirect `call` and `jmp` instructions to those entry points which were used indirectly in the assembly code. The indirect `call`/`jmp` protection could be improved if the programmer or higher-level language provided more precise insight into how indirect control flow transfers were to be used.

Second, our threat model presumes that the exploit in play can be used to alter the application's memory. The CFL technique is only intended to mitigate the risk of a code-reuse attack, which exploits the program's control data. However, as Chen et al. correctly observed, "Non-control-data attacks are realistic threats" [38]. That is, malicious attacks on plain variables can yield significant security problems, including unauthorized access and privilege escalation. Further, one can easily imagine a scenario in which a pure-data attack could even lead to malicious Turing complete behavior without altering control flow. Interpreters are a straightforward example of this risk: overwriting the "code" being interpreted yields arbitrary attacker-controlled behavior. As such, the CFL technique is not a *silver bullet* for all the dangers posed by software exploits; it mitigates the threat posed specifically by code-reuse attacks.

The implementation presented in Section 4 was based on statically linked binaries. This was primarily for ease of implementation; there is no reason in principle why the technique could not be applied to dynamically linked binaries. From a conceptual perspective, the conversion is straightforward: (1) the analysis of the call graph currently performed in the post-ld phase would be moved to the runtime linker, and (2) because we do not know at build time which exported symbols may be called indirectly, unlock code would be inserted in all exported functions' entry points, to be removed as needed by the runtime linker. Of course, the call graph of different programs would lead to different `k` values within dynamic shared libraries, which is problematic, as the library code pages would no longer be able to be shared. One possible solution would be to add a layer of indirection to the lookup of `k` values: instead of embedding the value directly in the instruction, a lookup into a per-process table could be substituted. It is not immediately clear what the additional overhead of this modification would be and we leave it to our future work.

The G-Free technique, which modifies assembly code to remove unintended indirect control flow transfers, may present an interesting extension to CFL. Currently, the problem of unintended code is solved by imposing alignment based on prior work on sandboxing [35, 36]. However, it may be possible to replace this technique with the branch-removal algorithm employed by G-Free, while leaving the actual control flow lock/unlock code as-is. This hybrid technique may have performance benefits which could make it more attractive than the current G-Free or CFL systems. Examining this possibility is another research question for us to explore.

7. CONCLUSION

This paper presented a novel defense against code-reuse attacks called *control flow locking* (CFL). This technique ensures that the control flow graph of an application is deviated from no more than once, and that this deviation cannot be used to craft a malicious system call. CFL works by performing a "lock" operation before each indirect control flow transfer, with a corresponding "unlock" operation present at valid destinations only. The technique has been implemented in practice on a commodity x86 system, and it has been shown to offer performance overhead competitive with existing techniques, achieving significant gains in several benchmarks. CFL represents a step further to mitigate code-reuse attacks with a small performance penalty.

8. REFERENCES

[1] Wikipedia. W^X. http://en.wikipedia.org/wiki/W^X.
[2] Solar Designer. Getting around non-executable stack (and fix). *Bugtraq*, 1997.
[3] Hovav Shacham. The Geometry of Innocent Flesh on the Bone: Return-into-libc without Function Calls (on the x86). In *14th ACM CCS*, 2007.
[4] Ping Chen, Hai Xiao, Xiaobin Shen, Xinchun Yin, Bing Mao, and Li Xie. Drop: Detecting return-oriented programming malicious code. In *5th ACM ICISS*, 2009.

[5] Lucas Davi, Ahmad-Reza Sadeghi, and Marcel Winandy. Dynamic Integrity Measurement and Attestation: Towards Defense against Return-oriented Programming Attacks. In *4th ACM STC*, 2009.

[6] Jinku Li, Zhi Wang, Xuxian Jiang, Mike Grace, and Sina Bahram. Defeating return-oriented rootkits with return-less kernels. In *5th ACM SIGOPS EuroSys Conference*, April 2010.

[7] Lucas Davi, Ahmad-Reza Sadeghi, and Marcel Winandy. ROPdefender: A detection tool to defend against return-oriented programming attacks. Technical Report HGI-TR-2010-001, Horst Görtz Institute for IT Security, March 2010.

[8] Tzi-cker Chiueh and Fu-Hau Hsu. RAD: A Compile-Time Solution to Buffer Overflow Attacks. In *21st IEEE ICDCS*, April 2001.

[9] Mike Frantzen and Mike Shuey. StackGhost: Hardware Facilitated Stack Protection. In *10th USENIX Security Symposium*, 2001.

[10] Vendicator. Stack Shield: A "Stack Smashing" Technique Protection Tool for Linux. *http://www.angelfire.com/sk/stackshield/info.html*.

[11] Tyler Bletsch, Xuxian Jiang, Vince Freeh, and Zhenkai Liang. Jump-Oriented Programming: A New Class of Code-Reuse Attack. In *6th AsiaCCS*, March 2011.

[12] Stephen Checkoway, Lucas Davi, Alexandra Dmitrienko, Ahmad-Reza Sadeghi, Hovav Shacham, and Marcel Winandy. Return-Oriented Programming Without Returns. In *17th ACM CCS*, October 2010.

[13] Martín Abadi, Mihai Budiu, Úlfar Erilingsson, and Jay Ligatti. Control-Flow Integrity: Principles, Implementations, and Applications. In *12th ACM CCS*, October 2005.

[14] Nergal. The Advanced Return-into-lib(c) Exploits: PaX Case Study. *Phrack Magazine, Volume 11, Issue 0x58, File 4 of 14*, December 2001.

[15] Erik Buchanan, Ryan Roemer, Hovav Shacham, and Stefan Savage. When Good Instructions Go Bad: Generalizing Return-Oriented Programming to RISC. In *15th ACM CCS*, pages 27–38, New York, NY, USA, 2008. ACM.

[16] Felix "FX" Lidner. Developments in Cisco IOS Forensics. In *CONference 2.0*, November 2009.

[17] Tim Kornau. *Return oriented programming for the ARM architecture*. Master's thesis, Ruhr-Universität Bochum, January 2010.

[18] Stephen Checkoway, Ariel J. Feldman, Brian Kantor, J. Alex Halderman, Edward W. Felten, and Hovav Shacham. Can DREs provide long-lasting security? The case of return-oriented programming and the AVC Advantage. In *EVT/WOTE 2009, USENIX*, August 2009.

[19] Stefan Esser. Utilizing Code Reuse/ROP in PHP Application Exploits. In *BlackHat USA*, 2010.

[20] Ralf Hund, Thorsten Holz, and Felix C. Freiling. Return-oriented rootkits: Bypassing kernel code integrity protection mechanisms. In *19th USENIX Security Symposium*, August 2009.

[21] Daniele Perito Claude Castelluccia, Aurélien Francillon and Claudio Soriente. On the Difficulty of Software-Based Attestation of Embedded Devices. In *16th ACM CCS*, New York, NY, USA, 2009. ACM.

[22] Crispin Cowan, Calton Pu, Dave Maier, Heather Hintony, Jonathan Walpole, Peat Bakke, Steve Beattie, Aaron Grier, Perry Wagle, and Qian Zhang. StackGuard: automatic adaptive detection and prevention of buffer-overflow attacks. In *7th USENIX Security*, page 5, 1998.

[23] PaX Team. PaX ASLR Documentation. http://pax.grsecurity.net/docs/aslr.txt.

[24] Sandeep Bhatkar, Daniel C. DuVarney, and R. Sekar. Address Obfuscation: An Efficient Approach to Combat a Broad Range of Memory Error Exploits. *12th USENIX Security*, 2003.

[25] Sandeep Bhatkar, R. Sekar, and Daniel C. DuVarney. Efficient Techniques for Comprehensive Protection from Memory Error Exploits. *14th USENIX Security*, 2005.

[26] Jun Xu, Zbigniew Kalbarczyk, and Ravishankar K. Iyer. Transparent Runtime Randomization for Security. *22nd SRDS*, October 2003.

[27] Tyler Durden. Bypassing PaX ASLR Protection. *Phrack Magazine, Volume 11, Issue 0x59, File 9 of 18*, June 2002.

[28] Hovav Shacham, Matthew Page, Ben Pfaff, Eu-Jin Goh, Nagendra Modadugu, and Dan Boneh. On the Effectiveness of Address Space Randomization. *11th ACM CCS*, 2004.

[29] Vladimir Kiriansky, Derek Bruening, and Saman Amarasinghe. Secure Execution Via Program Shepherding. In *11th USENIX Security Symposium*, August 2002.

[30] Miguel Castro, Manuel Costa, and Tim Harris. Securing Software by Enforcing Data-Flow Integrity. In *7th USENIX OSDI*, November 2006.

[31] Úlfar Erlingsson, Martin Abadi, Michael Vrable, Mihai Budiu, and George C. Necula. XFI: Software Guards for System Address Spaces. In *7th USENIX OSDI*, 2006.

[32] Periklis Akritidis, Cristian Cadar, Costin Raiciu, Manuel Costa, and Miguel Castro. Preventing Memory Error Exploits with WIT. In *28th IEEE Symposium on Security and Privacy*, May 2008.

[33] Miguel Castro, Manuel Costa, Jean-Philippe Martin, Marcus Peinado, Periklis Akritidis, Austin Donnelly, Paul Barham, and Richard Black. Fast Byte-Granularity Software Fault Isolation. In *22nd ACM SOSP*, October 2009.

[34] Kaan Onarlioglu, Leyla Bilge, Andrea Lanzi, Davide Balzarotti, and Engin Kirda. G-free: Defeating return-oriented programming through gadget-less binaries. In *ACSAC*, 2010.

[35] Stephen McCamant and Greg Morrisett. Efficient, verifiable binary sandboxing for a CISC architecture. In *MIT Tech Report MIT-CSAIL-TR-2005-030*, 2005.

[36] Bennet Yee, David Sehr, Gregory Dardyk, J. Bradley Chen, Robert Muth, Tavis Ormandy, Shiki Okasaka, Neha Narula, and Nicholas Fullagar. Native Client: A sandbox for portable, untrusted x86 native code. *Communications of the ACM*, 53(1):91–99, 2010.

[37] Felix von Leitner et al. dietlibc. http://www.fefe.de/dietlibc/.

[38] Shuo Chen, Jun Xu, Emre C. Sezer, Prachi Gauriar, and Ravishankar K. Iyer. Non-control-data attacks are realistic threats. In *14th USENIX Security*, pages 177–192, 2005.

deRop: Removing Return-Oriented Programming from Malware

Kangjie Lu
Peking University, China
Singapore Management University, Singapore
kangjielu@pku.edu.cn

Dabi Zou
Singapore Management University, Singapore
zoudabi@gmail.com

Weiping Wen
Peking University, China
weipingwen@ss.pku.edu.cn

Debin Gao
Singapore Management University, Singapore
dbgao@smu.edu.sg

ABSTRACT

Over the last few years, malware analysis has been one of the hottest areas in security research. Many techniques and tools have been developed to assist in automatic analysis of malware. This ranges from basic tools like disassemblers and decompilers, to static and dynamic tools that analyze malware behaviors, to automatic malware clustering and classification techniques, to virtualization technologies to assist malware analysis, to signature- and anomaly-based malware detection, and many others. However, most of these techniques and tools would not work on new attacking techniques, e.g., attacks that use return-oriented programming (ROP).

In this paper, we look into the possibility of enabling existing defense technologies designed for normal malware to cope with malware using return-oriented programming. We discuss difficulties in removing ROP from malware, and design and implement an automatic converter, called deRop, that converts an ROP exploit into shellcode that is semantically equivalent with the original ROP exploit but does not use ROP, which could then be analyzed by existing malware defense technologies. We apply deRop on four real ROP malwares and demonstrate success in using deRop for the automatic conversion. We further discuss applicability and limitations of deRop.

Keywords: return-oriented programming, malware analysis

1. INTRODUCTION

Malware analysis has been one of the hottest areas in security research in the last 10 to 20 years. Many techniques and tools have been introduced and built to automatically analyze and defend against malware. This ranges from basic tools like disassemblers (e.g., IDAPro[1], OllyDbg[2]) and decompilers [11], to static [10, 16] and dynamic tools [4, 31] to analyze malware behaviors, to automatic malware clustering [3, 18] and classification techniques [22, 1], to virtualization techniques for analyzing malware [13], to signature- and anomaly-base malware detection [20, 27, 17, 30], and many others.

Although some of these techniques are designed to be able to deal with zero-day malware, e.g., anomaly-based detection, the vast majority of them are based on understanding of the existing malware techniques. Even anomaly-based detection might not work when the technology used by malware changes (instead of new malware exploiting an unknown vulnerability). Return-oriented programming is an example of such new malware technology.

Return-oriented programming (ROP) [26, 28] and its variations [7, 6, 14, 19, 21, 8, 5] have been shown to be able to perform arbitrary computation without executing injected code. They execute machine instructions immediately prior to return (or return-like [7]) instructions within the existing program or library code. Since ROP does not execute any injected code, it circumvents most measures that try to prevent the execution of instructions from user-controlled memory, e.g., the W $\oplus$ X protection mechanism [25].

One solution to this problem is to design patches to all existing malware defense technologies so that they can cope with return-oriented programming. Although not entirely impossible, it is definitely not scalable due to the huge amount of existing defense work. Even if it can be done, it will become a nightmare when yet another new malware technology emerges.

Instead, we propose to automatically remove return-oriented programming from a piece of malware before it is sent for further analysis by existing malware defense tools. In this paper, we design and implement deRop, an automatic tool to convert shellcode[3] using ROP into one that does not use

[1]http://www.hex-rays.com/idapro
[2]http://www.ollydbg.de
[3]The payload of an ROP attack usually contains many gadgets of addresses, constants, and junks. There is usually no instruction or code in an ROP payload, and strictly speaking we should not call it shellcode. Here we still use the word shellcode to refer to payload of an attack in general, should it use ROP or not.

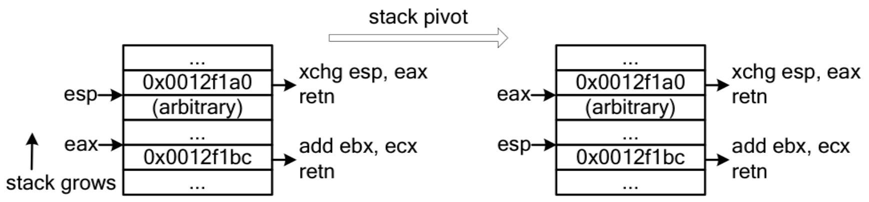


Figure 1: Special use of esp

ROP, while preserving the semantics of the original malware shellcode. Note that here we focus on enabling existing malware analysis tools to work on the output of deRop. deRop does not perform this analysis on the malware directly.

Removing return-oriented programming from malware is non-trivial. Among the many difficulties (discussed in Section 3), the use of register `esp` in ROP shellcode is one of the most important ones. Figure 1 shows an example in which `esp` is used as a stack pivot [2] to chain the execution of gadgets (address words pointing to "hidden" instructions and data) in ROP. The stack layouts on the left and right show the content of `esp` and `eax` when instructions pointed to by the two gadgets start getting executed, respectively. It is relatively easy to find out all the "hidden" instructions to be executed pointed to by the gadgets, however, deRop cannot simply take these instructions as the output of the conversion. For example, the instruction `<xchg esp, eax>` pointed to by one of the gadgets is used to update the value of `esp` so that it points to the next gadget to be executed. In this sense, `esp` in ROP serves the functionality of `eip`. `<xchg esp, eax>`, therefore, is not really part of the instructions this attack tries to execute, and should not be part of the output of deRop.

Another difficulty is to be able to find out the next gadget (and the corresponding "hidden" instructions). Intuitively, dynamic analysis could help solve this relatively easily. However, it comes with a price of having to execute the malware in the analysis, which is a big disadvantage considering the security of the analysis platform. Therefore, one of the criteria we have in designing deRop is to use static analysis to simulate the updates to `esp` as much as possible, and only resolve to dynamic analysis when needed.

We design and implement deRop to automatically convert shellcode using ROP into one that does not use ROP. deRop relies heavily on static analysis without executing the malware, with a minimum of dynamic analysis only to find the location of the first gadget in the malware payload and to find the initial value of `esp`, which are hard to obtain reliably with static analysis alone. We apply deRop on four real ROP malwares and manually verify that the output of deRop does not make use of ROP, and is semantically equivalent to the original ROP malware. We also discuss applicability of deRop and its limitations.

2. BACKGROUND AND RELATED WORK

2.1 ROP and its variations

Shacham et al. propose Return-Oriented Programming (ROP) in 2007 [28]. ROP uses a large number of instruction sequences from the original program and the libraries, and chains these instruction sequences ending with `ret` together to perform arbitrary computation. ROP is also ported to other platforms such as SPARC [6], ARM [21], Harvard [14], and voting machines [8]. Besides that, Hund et al. [19] make use of ROP to propose Return-Oriented Rootkits, which can bypass kernel code integrity protection mechanisms. Li et al. [23] propose the corresponding "return-less" kernels to defeat Return-Oriented Rootkits. However, ROP is recently extended to use return-less gadgets to achieve the same effect [7, 5].

2.2 Defense of ROP

With the development of ROP, some researches are seeking ways to detect and prevent ROP attacks. One direction is to make use of the characteristics of ROP, e.g., short pieces of instructions ended by a `ret`. Davi et al. [12] and Chen et al. [9] make use of dynamic binary instrumentation frameworks to instrument program code. When the number of consecutive sequences of five or fewer instructions ending in a return reaches a threshold, it will trigger an alarm. Another direction is to look for violations of last-in, first-out invariants of the stack data structure that call and return instructions usually maintain in normal benign programs. Buchanan et al. [6] suggest to maintain a shadow return-address stack, which can be used to defend against ROP. Francillon et al. [15] implement a shadow return-address stack in hardware for an Atmel AVR microcontroller such that only call and return instructions can modify the return-address stack.

Davi et al. [12] claim that it is possible to extend their ROP defender with a frequency measurement unit to detect attacks with return-less ROP. The idea is that pop-jump sequences are uncommon in ordinary programs, while return-less ROP [7] invokes such a sequence after each instruction sequence.

Most recently, Onarlioglu et al. [24] propose G-Free, which is a compiler-based approach to defeat against any possible form of ROP. Their solution is to eliminate all unaligned free-branch instructions inside a binary executable, and to prevent aligned free-branch instructions from being misused.

Unlike these defense mechanisms, deRop does not try to detect or stop ROP. Instead, it converts any (attack) program that uses ROP into one that does not. This has huge implication on the applicability of existing malware analysis tools and they could now be used to analyze ROP attacks.

2.3 Existing malware analysis tools

There have been many malware analysis tools proposed. Some examples include disassemblers (e.g. IDAPro, OllyDbg) and decompilers (e.g., HexRays decompiler[4]), static [10, 16] and dynamic tools [4, 31] to analyze malware behav-

[4]http://hex-rays.com/decompiler.shtml

iors, automatic malware clustering [3, 18] and classification techniques [22, 1], virtualization techniques for analyzing malware [13], signature- and anomaly-base malware detection [20, 27, 17, 30], and many others.

Most, if not all, of these techniques were proposed to analyze traditional malware without considering ROP. In fact, most of them were proposed before ROP was introduced. Traditional malware code (usually in form of shellcode) is very different from ROP in that ROP shellcode consists of addresses, constants and junks but not instructions. Therefore, direct applications of such tools on ROP code will likely fail.

In this paper, we are not trying to propose a new design of these malware analysis tools so that they could work on ROP. There are too many such tools proposed and it is not practical or scalable to patch all of them. Instead, we propose an automatic converter, called deRop, to convert ROP shellcode into its semantically equivalent non-ROP shellcode so that any of the existing malware analysis tools can analyze it.

3. DIFFICULTIES

In Section 1, we briefly discuss why existing malware analysis tools are not able to analyze ROP shellcode effectively. In this section, we detail the difficulties involved in designing deRop, which takes input some shellcode that uses ROP and outputs non-ROP shellcode to be analyzed by existing malware analysis tools.

3.1 Locations of gadgets

The payload of an ROP attack would usually leave some junk before the first gadget, e.g., if the exploit is via a buffer overflow. Therefore, deRop needs to find a way of locating the first gadget in the ROP shellcode. In some SEH exploits, there are even junks between the first and the second gadget since where `esp` points to might not follow the location of the return address. Although deRop tries to use static analysis as much as possible and tries to avoid using dynamic analysis (see discussion in Section 1), locating the first two gadgets in an ROP attack using static analysis turns out to be unreliable, and therefore we customize a debugger to do it instead. Note that the debugger never runs any malicious code.

3.2 Keeping track of esp

As pointed out in Section 1, `esp` in ROP shellcode has a special usage as a global state pointer whose function is to get the address of the next group of instructions, just like `eip` in normal programs. We need to keep track on the value of `esp`, e.g., to locate the next gadget to be used in the execution. This is non-trivial especially in sophisticated ROP shellcode that has conditional branches.

3.3 Stack layout and constant relocation

ROP and non-ROP code load constants into registers in very different ways. Traditional shellcode usually uses `<mov reg, imm>`, while ROP shellcode usually pre-arranges the constant on the stack, and then uses `<pop reg>` to load the constant into a register. Therefore, deRop needs to relocate the constants in its transformation, and the input and output of deRop has very different stack layout. A mapping between these constants in the input ROP shellcode and the output non-ROP shellcode is developed to keep track on them.

3.4 Function calls

Some gadgets in ROP are used to call functions. This is usually achieved by stack pivot instructions or `<pushad>` followed by `ret`. One may argue that no special treatment is needed for function calls as we can simply inline their body to remove ROP and preserve semantics at the same time. However, the objective of deRop is to enable other malware analyzers to be able to analyze the resulting non-ROP shellcode, therefore being able to recognize these function calls and to conform to normal function call conventions is important. Difficulties here include identifying gadgets that perform function calls and parameter usage (e.g., parameters being constants or pointers) for the call.

3.5 Loops

Loops in ROP are usually implemented using stack pivot to perform conditional jumps. The difficulty here inlcudes identifying loops as well as finding out the actual condition of the loop. deRop uses some heuristics to handle these difficulties, which turn out to be effective in our experiments.

4. DESIGN AND IMPLEMENTATION OF deRop

In this section, we detail the design and implementation of deRop. We first give an overview of the design in Section 4.1, and then present a running example to aid the explanation (see Section 4.2). Section 4.3, Section 4.4, and Section 4.5 describe the three main steps in the design of deRop. Finally, we briefly describe how deRop is implemented in Section 4.6.

4.1 Overview

Although converting ROP shellcode to non-ROP shellcode to be analyzed by existing malware analysis tools is non-trivial (see difficulties discussed in Section 3), we design and implement deRop, an automatic tool to remove return-oriented programming from a piece of code. Figure 2 shows an overview of the design of deRop.

deRop first uses a customized debugger to find out the locations of of the first two gadgets in the input ROP shellcode. This is the only component in deRop that uses dynamic analysis, and our special design of the debugger makes sure that potentially harmful instructions in the attack code do not get executed (we don't even take the ROP shellcode as input to the vulnerable program).

After finding out the locations of the first two gadgets in the ROP shellcode, deRop uses a loop to analyze each individual gadget. This step of the analysis employs only static analysis techniques to simulate the execution of each gadget, and then output instructions that do not use ROP. During the simulation, deRop keeps track of many important information including the register values, mapping between addresses of instructions and data in the ROP input shellcode and non-ROP output shellcode. Finally, deRop performs a post-processing to improve the output shellcode so that it can be readily analyzed by existing malware analyzers.

4.2 A running example

Due to the complexity of certain parts of deRop, we find it easier to understand if we explain it with a simple example.

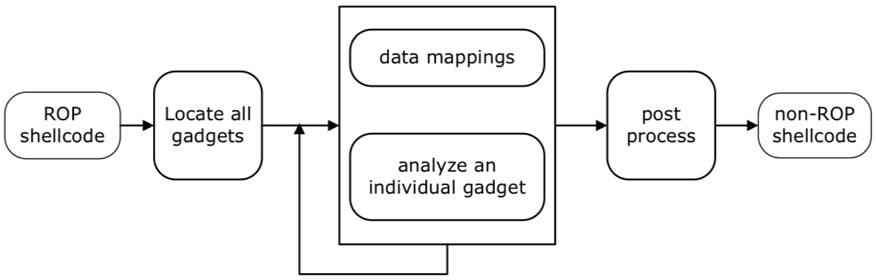


Figure 2: Overview of deRop

We use a real ROP exploit from `exploit-db`[5] which exploits CoolPlayer 2.18[6] with DEP (Data Execution Prevention) . The transferred results are shown in Table 1.

We choose this ROP exploit because it is simple and exhibits a typical structure of existing ROP exploits in that they first use ROP to perform a crucial step in the attack, and then trigger a piece of traditional (non-ROP) shellcode to perform whatever the attacker wants to achieve. In this particular example, the ROP part uses the function `SetProcessDEPPolicy()` to disable DEP for the process, so that the traditional shellcode in the data segment can be executed.

The first three columns of Table 1 show the structure of this exploit, where the first part contains junks to perform buffer overflow, the second part uses 7 gadgets (the first gadget doesn't play a role in this attack) to perform ROP, and the last part contains the traditional non-ROP shellcode to be executed. Gadget 2 to 6 are used to set up register values to be pushed on the stack by gadget 7 using `pushad`. The subsequent `ret` instruction causes the process to look up a return address on the stack (which is located where `esi` is pushed) and to transfer control to that location.

The stack layout right after `pushad` is executed is shown in Figure 3. Gadget 6 and 5 set `esi` and `edi` as pointers to `ret` instructions, which means that when control is transferred to them, they do nothing but moves on. `ebp` is set to be the address of `SetProcessDEPPolicy()` by gadget 4, which, when called, uses the value of `ebx` as its parameter. Gadget 2 and 3 set `ebx` to be 0, so that when `SetProcessDEPPolicy(0)` is called, data execution prevention is disabled.

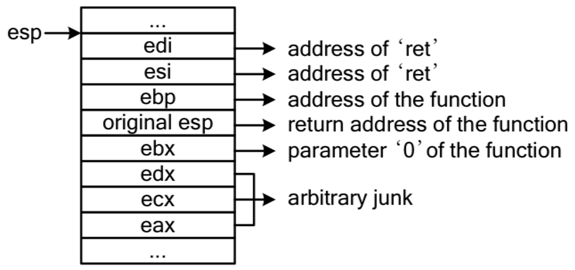


Figure 3: Stack layout right after pushad is executed in our example

4.3 Locating gadgets

The first challenge we face is to locate the gadgets in an ROP exploit code. As discussed in Section 3 and shown in Table 1, this is non-trivial since the ROP exploit may contain junks before the first gadget. In some cases, there are even junks between the first and the second gadgets. To reliably find out the locations of the first and the second gadgets, we design a debugger to dynamically monitor the execution of the vulnerable program.

We stress that this is the only component of deRop that does dynamic analysis, and deRop does it in such a way that potentially harmful instructions in the attack code are never executed.

We first prepare a buffer that is of the same length of the ROP exploit as shown in Figure 4 to be used to exploit the vulnerable program. There are two index numbers in this buffer — a byte-index number and a word-index number denoted $\mathsf{idx_b}(\mathsf{x})$ and $\mathsf{idx_w}(\mathsf{x})$, respectively, where x is a 4-byte word in the buffer. $\mathsf{idx_b}(\mathsf{x})$ (2 bits long) appears in each byte of x, and is used to find the alignment offset in case the gadgets in the exploit are not aligned at multiples of 4-byte words. $\mathsf{idx_w}(\mathsf{x})$ (24 bits long) is used to tell which 4-byte word the first gadget is located[7]. We execute the vulnerable program, use this buffer to exploit it, and observe the values of `eip` and `esp` when the exploit succeeds. The value of `eip` tells us the size of the junk before the first gadget, while the value of `esp` tells us the size of the junk between the first two gadgets (indirectly).

When testing with the running example shown in Table 1, our debugger finds that $\mathsf{idx_w}(\mathsf{eip}) = 55$, $\mathsf{idx_w}(\mathsf{esp}) = 56$, and $\mathsf{idx_b}(\mathsf{eip}) = \mathsf{idx_b}(\mathsf{esp}) = 0$. This means that location of the first gadget is at an offset of $55 \times 4 + 0 = 220$ bytes, and the value of `esp` is at an offset of $56 \times 4 + 0 = 224$ bytes before the first gadget executes.

Now we have managed to find out the location of the first gadget without executing any potentially harmful instructions in the ROP exploit. The next is to find out the location of the second gadget. As explained in Section 1, `esp` plays the role of `eip` in ROP. Therefore, all we need is to find out the value of `esp` *after* the execution of (instructions pointed to by) the first gadget. This can be achieved by analyzing how such instructions change the value of `esp`, since we already know the value of `esp` *before* the execution of the first gadget.

In our running example, the first gadget points to the instruction `pop ecx` (see Table 1), which added 4 to the

[5]`http://www.exploit-db.com/`
[6]`http://www.exploit-db.com/exploits/15895/`

[7]Note that the 8th bit is 1 instead of 0 in order to make sure that none of the bytes in the buffer has a value of 0.

Table 1: A running example

ROP shellcode			Resulting non-ROP shellcode	
Gadget #	Payload	Instructions	Initial results	Post-process
	"0x41" ×220	`junk`		

1	0x7c9fb028
	0x42424242
		`pop ecx`
		`retn`
			`mov ecx,[0x12f1ab]`	`mov ecx,0x42424242`

2	0x7c9eaddd
	0xffffffff
		`pop ebx`
		`retn`
			`mov ebx,[0x12f1af]`	`mov ebx,0xffffffff`

3	0x77c127e1	`inc ebx`
		`retn`
			`inc ebx`	`inc ebx`

4	0x7c9ea67b
	0x7c8922a4
		`pop ebp`
		`retn`
			`mov ebp, [0x12f1b3]`	`mov ebp, 0x7c8922a4`

5	0x7c9eeb47
	0x7c9c1508
		`pop edi`
		`retn`
			`mov edi, [0x12f1b7]`	`mov edi, 0x7c9c1508`

6	0x7c9c204c
	0x7c9c2051
		`pop esi`
		`retn`
			`mov esi, [0x12f1bb]`	`mov esi, 0x7c9c2051`

7	0x7ca11073	`pushad`
		`retn`
			`mov esp, 0x12f1bf`
			`pushad`
			`pop edi`
			`jmp edi`
				`mov esp, 0x12f1ab`
				`pushad`
				`pop edi`
				`jmp edi`

	"0x90" ×10 240 bytes	`nop` `shellcode`	0x42424242 0xffffffff 0x7c8922a4 0x7c9c1508 0x7c9c2051	nil

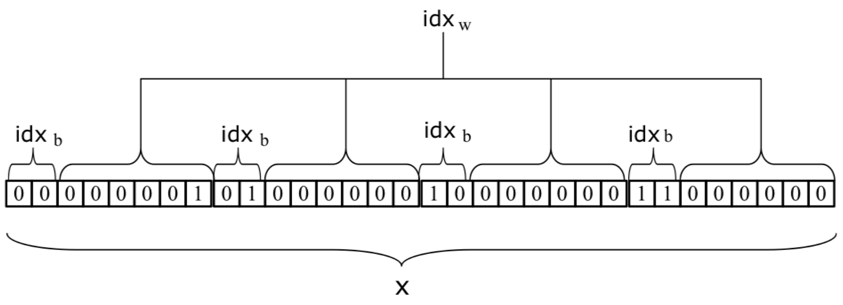


Figure 4: Debugger payload format

value of `esp`. Therefore, we know that `esp` is at an offset of $224 + 4 = 228$ bytes, which will also be the location of the second gadget. Note that this means that there is no junk between the first two gadgets, since the size of the first gadget is 8, which is exactly the difference between the offsets of the first two gadgets.

Locating subsequent gadgets follows the same idea by monitoring the value of `esp`. Note that dynamic analysis is not needed in locating subsequent gadgets, since all we need is the change to `esp` as the result of executing (instructions pointed to by) the gadgets, which can be obtained by static analysis.

4.4 Removing ROP

As discussed in Section 3, removing ROP from an exploit is not as simple as chaining all instructions pointed to by the gadgets. We need to analyze the instructions pointed to by each gadget one by one, and monitor 1) address of each instruction, 2) values of registers, and 3) addresses of memory accesses. This is very similar to simulating the execution of the ROP shellcode, although we perform all this statically while preserving the semantics of the original ROP shellcode. Next, we detail how deRop deals with various types of instructions pointed to by each gadget.

4.4.1 Push and pop instructions

`push` and `pop` instructions are used very often in ROP. In particular, they are used together to transfer the value of one register to another, the latter typically being `esp`. ROP also usually prepares constants in its payload, which are on the stack during the initial exploit, and pops these constants from the stack to various registers. A few examples are shown in Table 1 in gadgets 1, 2, 4, 5, and 6.

Table 2 shows how deRop deals with `push` and `pop` instructions. In cases where `push` and `pop` are used together (see the second row of Table 2), we use a `mov` instruction to replace them, where `addr1` is the location of the data that is used to initialize `eax`. Note that here we do not replace them with `<mov ecx, eax>`. The reason is that `pop` does not clear the data (of `eax`) on the stack, and therefore such data might be used again later in the ROP exploit. When `pop` is used alone to assign a constant prepared on the stack to a register, we replace `pop` with the corresponding `mov` instruction as shown in Table 2.

An important requirement to enable this transformation from ROP instructions to non-ROP instructions is that we keep track on the location of data in memory. For example, `addr1` is the address of the data that is stored in `eax`, and `addr2` is the address of the data the ROP originally prepares

Table 2: Push and pop transferring example

ROP instructions	non-ROP instructions
`push eax; pop ecx`	`mov ecx, [addr1]`
`pop ecx`	`mov ecx, [addr2]`

on the stack. deRop keeps track on the addresses of them in the process of analyzing the instructions.

Another challenge is that the memory layout of the ROP exploit and the resulting non-ROP exploit might not be the same. In our example shown in Table 1, the gadgets in the payload contains constants, which is unique in ROP. Non-ROP shellcode does not usually mix code and data together in such a special way (it may do so with immediates as operands of instructions, which will be discussed in Section 4.5). Therefore, deRop relocates such data to another location in memory, e.g., see the last 20 bytes of the initial results presented in Table 1.

4.4.2 Memory related instructions

deRop deals with memory related instructions by setting aside a memory region in the output of deRop to store constants, and keeping a mapping between the corresponding constants in ROP and non-ROP shellcode. As shown in Table 1, the non-ROP shellcode reserves the last few bytes to store 5 constants, which were originally located separately in 5 different gadgets.

Table 3 shows two examples of instructions that involve memory access. In the first example, the address `0x1234` is outside the range of the ROP exploit code. For these type of addresses, deRop keeps them unchanged. While in the second example, the address `0x12345678` falls in the range of the original ROP shellcode, and deRop replaces it with a new memory location at the end of the resulting non-ROP shellcode (`addr3`). deRop updates its mapping of the two locations of this data so that future access of the same data can be handled properly.

Table 3: Memory related instruction transferring example

ROP instructions	non-ROP instructions
`add ecx,[0x1234]`	`add ecx,[0x1234]`
`add [0x12345678], ecx`	`add [addr3], ecx`

4.4.3 Stack pivot instructions

ROP shellcode typically uses stack pivot instructions to set the value of `esp`. This is critical in ROP since `esp` has a special usage as a global state pointer (just like `eip`) to get the address of the next gadget.

deRop monitors all the stack pivot instructions (e.g., `<add esp, 8>`; `<xchg esp, eax>`) to monitor the value of `esp`. Note that deRop simply needs to monitor it so that it knows where the next gadget is, while there does not need to have any corresponding instructions in the output of deRop, see the first example in Table 4.

Table 4: Stack pivot transferring example

ROP instructions	non-ROP instructions
`add esp, 20`	nil
`xchg eax, esp`	`cmp eax, addr4; jz offset1`

However, there is a special case we need to be careful of, which is when `esp` points to a gadget that has been analyzed by deRop. This happens in some advanced ROP shellcode that does looping. deRop recognizes this by checking the value of `esp` against addresses of all gadgets, and handles it with a conditional jump instruction in the output.

In the second example shown in Table 4, deRop finds out that `esp = addr4` points to a gadget previously analyzed that is located at an offset of `offset1`. In this case, the condition of the loop in the original ROP shellcode is represented by the value of `eax`, which equals to `addr4` except in the last round of the loop. Therefore, we compare the value of `eax` and `addr4` in the non-ROP shellcode, and continues the loop by jumping to offset `offset1`. Note that we apply the same strategy in other cases where it is suspected that a conditional jump is intended.

4.4.4 Function calls

Function calls are common in ROP. Special handling of function calls is not a necessity because as long as all instructions in the function are converted with semantics preserved, deRop is sound. In the example shown in Table 1, we could have continued our analysis after gadget 7, after which control is transferred to the function `SetProcessDEPPolicy()`. In that case we will be using deRop to analyze the body of `SetProcessDEPPolicy()`, which is unnecessary (since this function does not use ROP) and degrades readability (the output of deRop will not show a function call but with the body inlined). Due to these disadvantages, we decide to make an effort to recognize function calls and follow some function call conventions in the output of deRop.

Recognizing function calls in ROP can be done by identifying some function characteristics of the epilogue and prologue. For example, the beginning of a function usually contains instructions to change the value of `ebp`, `esp`, etc. However, such characteristics are not reliable especially in release versions of libraries, and we choose not to rely on them.

Instead, we use a simple but reliable heuristic that the number of instructions in a function is usually much larger than that in instructions pointed by gadgets in ROP. In particular, the number of instructions pointed to by a gadget is usually smaller than 5, while there are usually more than 50 instructions in a function. We therefore set a threshold of 50 in recognizing functions. Even if we fail to identify some very small functions using this heuristic, the drawback is minimal as not following function call convention for such a small function would not have seriously affected the malware analyzer's performance. Another heuristic deRop uses is to match the destination address with those in the export table (Windows) or GOT/PLT (Linux). If the address exists in the tables, it obviously corresponds to a library function call.

After identifying a function call, deRop chains the parameters, variables, and the return address on the stack in the right order and then updates `esp` to point to the first parameter. In cases where the parameters or variables are pointers, deRop also prepares the corresponding data and structure pointed to by these parameters and variables.

In the example shown in Table 1, there are two things to do to make sure that the function call works well in the resulting non-ROP shellcode. One is to set up `esp` as the return address after the function returns. This is simply the current value of `esp` in the non-ROP shellcode, which will be the value of `eip` when the non-ROP shellcode is executed.

The other is to set up `edi` as the target of the jump, which is the address of the function to be called. We use `jmp` instead of `call` instruction because the execution of `call` pushes the address of the next instruction on the stack, which diverges the ROP execution.

4.4.5 *Unconditional jump instructions*

An ROP exploit might use unconditional jumps to execute some instruction sequences indirectly. For example, the gadget may point to `<jmp eax>` while `eax` points to the instructions to be executed. In this case, deRop simply recursively applies the analyzing process on the instructions at the jump target.

4.4.6 *Other instructions*

These instructions can be processed easily since they do not involve reading/writing of memory locations, changing of stack layout, or changing of control flow. We simply copy these instructions to the output of deRop. Examples include `<xor eax, ebx>`, `<add eax, ebx>`, and many others.

4.5 Post-processing

There are a few steps that we can perform after all gadgets are analyzed. The purpose of this post-processing is to make the output of deRop look more similar to traditional non-ROP shellcode.

4.5.1 *Data in memory*

As discussed in Section 4.4, we relocate some data/constant in the original ROP shellcode to a specific location in the output of deRop, e.g., the last 20 bytes of the initial results presented in Table 1. This output is correct in the sense that it is semantically equivalent to the original ROP, but traditional non-ROP shellcode might not do this, e.g., the constants are usually directly inserted into code as operands instead of being stored at a different location on the stack. Moreover, using a specific location in memory makes the output of deRop dependent on a specific execution instance. For example, such non-ROP shellcode might not execute well in different running instances of an ASLR [29, 32] system.

deRop performs its post-processing to find out the use of such data throughout the program/malware. If the data is not directly used as a parameter of a function to be called, and is not being accessed more than once, then deRop replaces the memory access with the immediate value in the operand of the instruction, and also deletes the data from its allocated memory location in the initial result[8]. In our example shown in Table 1, this applies to gadget 1, 2, 4, 5, and 6. We can see that all the 20 bytes at the end of the non-ROP shellcode are gone after post-processing, and the memory accesses are replaced by immediate values in the instructions. Under ASLR, we set it as an option for deRop to change the absolute addresses into relative addresses (i.e., offsets from the base of the resulting non-ROP shellcode).

4.5.2 *Null-bytes*

Exploit payloads usually require that null-bytes do not exist, since it truncates the payload in various operations. deRop first encodes all null-bytes into non-zero values, and then adds a decoder to the final shellcode to be executed at the beginning of the exploit to restore the original value.

4.5.3 *Return address*

In cases where the exploit payload is to overflow a buffer to overwrite the return address, deRop replaces the word that overwrites the return address with the address of the start of the resulting shellcode.

Note that the processing of null-bytes and return address are needed only if the output of deRop is used directly to exploit the vulnerable program.

4.6 Implementation

We have implemented deRop in C++ with Visual Studio 6.0 with less then 2,000 lines of code. deRop now is implemented as a prototype, but using deRop as a product (e.g., running deRop on end user machines) is just a engineering problem. We believe deRop can be fitted into large system for malware detection and analysis. It consists of two components, a debugger and an analyzer. The debugger takes as input the length of the original ROP shellcode and the vulnerable application, and outputs the offsets of the first gadget and the value of `esp` when control is first transferred to the exploit code. It uses some Windows API (e.g., `WaitForDebugEvent()`, `GetThreadContext()` and `ContinueDebugEvent()`) to get debug information and constructs a special buffer which is shown in Figure 4.

The analyzer takes input the information from the debugger as well as the original ROP shellcode, removes ROP from it and outputs the non-ROP shellcode. The implementation of it uses a third-party tool `ndisasm`[9] to disassemble the binary code pointed to by addresses in the gadgets.

With the implementation of deRop, we run some ROP exploit examples for evaluation, and show our results in Section 5.

5. EVALUATION

In this section, we report success in applying deRop to four real-world ROP exploits. In all four cases, deRop manages to remove ROP and output the semantically equivalent non-ROP shellcode. We use the resulting non-ROP shellcode to exploit the vulnerable programs, and confirm that the same behaviors are observed in the original ROP exploit and the non-ROP exploit.

The first two columns of Table 5 summarize the four vulnerable applications and the corresponding exploits we use in our evaluation. The exploits are all published at Exploit Database[10] and tested on Windows XP SP3. Note that a common feature in these ROP exploits is that they use ROP to call `SetProcessDEPPolicy()` to make the appended non-ROP shellcode executable. Some of them do this by calling `VirtualProtect()`. In this sense, these four examples are relatively easy as the ROP portion performs limited functionality. We stress that we choose these four examples not because of their simplicity. It is simply because the fact that most current real-world ROP exploits use this strategy.

The last three columns of Table 5 show the result of our debugger in locating the gadgets (see Section 4.3). Note that in two of the exploits the first gadget is located at a large offset, and deRop manages to find it.

[8]In cases where pushad is used to prepare data (parameters, variables, and return address) of a function call, deRop also replaces them with the corresponding immediate values.

[9]http://www.nasm.us/doc/nasmdoca.html

[10]`http://www.exploit-db.com`

Table 5: Locating gadgets in ROP exploits

Vulnerable application	Exploit	Offset of 1st gadget	Offset of `esp`	Value of `esp`
CoolPlayer 2.18	DEP bypass	212 bytes	216 bytes	0x12F18C
WM Downloader 3.1.2.2	Buffer overflow & DEP bypass	17,432 bytes	17,440 bytes	0x0DC5C0
MP3-Nator	Buffer overflow & SEH-DEP bypass	28 bytes	32 bytes	0x12FBB8
SnackAmp	Buffer overflow & SEH-DEP bypass	10,564 bytes	10,570 bytes	0x12FA80

5.1 Results of removing ROP from four real-world exploits

Table 6 shows the results of removing ROP from these four exploits. An interesting observation is that the output non-ROP shellcode and the original ROP shellcode are of about the same size. This is a bit counter-intuitive since each gadget in ROP points to a sequence of instructions, which is supposed to be longer than the gadget itself. Our investigation into the details reveal that this is because

- the instruction sequence pointed to by each gadget usually contains only less than 5 instructions;
- there is a lot of junk among gadgets in ROP;
- most of the `push`, `pop`, and `esp`-related instructions are removed in the output non-ROP shellcode.

5.2 Other observations and discussions

We also encounter some special situations when applying deRop on these real-world ROP exploits.

5.2.1 Non-ROP shellcode in ROP exploits

This is the case for most real-world ROP exploits. ROP first calls a function which makes the non-ROP shellcode executable, and then transfers control to the non-ROP shellcode. This shows the importance of deRop being able to recognize function calls in ROP so that the output of deRop can be easily analyzed by existing malware analysis tools.

5.2.2 System call

ROP can be used to make system calls, which is common on Linux. deRop is able to address this in the same way of handling functions as discussed in Section 4.4. deRop recognizes system call making by looking for `<call *%gs:0x10>`, which is the new system call instruction on Linux.

5.2.3 ASLR

The address of the stack is different in different running instances of the same program under address space randomization. deRop is capable of removing ROP from exploits that execute on ASLR systems. Note that due to the post-processing of data in memory as discussed in Section 4.5, the output of deRop is also independent of any specific running instance. That is, although the output of the dynamic analysis of deRop (value of `esp`) differs in each running instance, the output of deRop can be always the same, and is semantically equivalent to the original ROP and executes on the vulnerable application in all running instances.

5.2.4 Return-less ROP

While transferring ROP into non-ROP, `ret` instruction has two special functions: 1) indicating the end of the instruction sequences pointed by gadget; 2) changing the value of `esp`. These two functions are used to extract instruction sequences pointed by gadget and trace the value of `esp`. When come to return-less ROP, through it does not use `ret` instruction sequences ending with `ret`, there are also similar instructions act like `ret`, e.g., `jmp` instruction and `pop-jmp` instructions are used to chain next gadget. Thus, we can simply monitor the ret-like instructions rather than `ret` instruction to apply deRop in return-less ROP.

5.2.5 Semantically equivalent instructions

As discussed in Section 4.4, for different kinds of special instructions, we use corresponding instructions to replace them. Based on the characteristics of ROP and stack , we select the most direct and uniform instructions which are certain. Actually, for different ROP shellcode, the transferring processes are a little different and there are multiple semantically equivalent instructions, so deRop provides the post-processing to optimize the resulting non-ROP shellcode. The post-processing described in Section 4.5 is just to optimize the memory related instructions. There may be more direct and simpler but equivalent instructions to optimize other types of instructions, which related to equivalent semantic technique and we leave it for future work.

6. CONCLUSION AND LIMITATIONS

In this paper, we design and implement deRop to remove return-oriented programming from malware instances. deRop enables malware analyzers to use many existing malware analysis tools to analyze ROP-based malware, which has not been taken into consideration when the existing analysis tools were designed and built. deRop is a fully automated tool that preserves the semantics of the original malware. We evaluate deRop by applying it to four real-world ROP exploits and demonstrate its success in removing ROP and preserving semantics.

We have discussed some of the limitations of deRop in previous sections, e.g., its output is one running instance specific in ASLR. Besides that, deRop needs dynamically executing the vulnerable application in order to locate the gadgets in the ROP exploit. However, we stress that this dynamic analysis does not involve running any potentially harmful instructions in the original ROP exploit code. One last limitation of deRop is that its output might still be slightly different from traditional shellcode even with the post-processing. For example, the output of deRop calls a function using `jmp`.

7. REFERENCES

[1] M. Bailey, J. Oberheide, J. Andersen, Z. M. Mao, F. Jahanian, and J. Nazario. Automated classification and analysis of internet malware. In *Proceedings of the 10th International Conference on Recent Advances in Intrusion Detection (RAID)*, 2007.

[2] P. Bania. Security mitigations for return-oriented programming attacks. *Whitepaper, Kryptos Logic Research*, 2010.

Table 6: Results in removing ROP from four real-world ROP exploits

Exploit on	CoolPlayer		WM Downloader		MP3-Nator		SnackAmp	
Code	ROP	non-ROP	ROP	non-ROP	ROP	non-ROP	ROP	non-ROP
Size (bytes)	488	495	21,831	21,418	1,003	1,202	16,700	16,764
# of gadgets	7	N/A	44	N/A	34	N/A	53	N/A
# of instructions	N/A	10	N/A	80	N/A	50	N/A	86
# of instructions executed	16	10	134	80	91	50	133	86

[3] U. Bayer, P. M. Comparetti, C. Hlauschek, C. Krugel, and E. Kirda. Scalable, behavior-based malware clustering. In *Proceedings of the 16th Annual Network and Distributed System Security Symposium (NDSS)*, 2009.

[4] U. Bayer, A. Moser, C. Krugel, and E. Kirda. Dynamic analysis of malicious code. *Journal in Computer Virology, Volume 2(1), 67-77*, 2006.

[5] T. Bletsch, X. Jiang, and V. W. Freeh. Jump-oriented programming: A new class of code-reuse attack. In *Proceedings of the 6th ACM Symposium on Information, Computer and Communications Security (ASIA CCS)*, 2011.

[6] E. Buchanan, R. Roemer, H. Shacham, and S. Savage. When good instructions go bad: generalizing return-oriented programming to risc. In *Proceedings of the 15th ACM conference on Computer and communications security (CCS)*, 2008.

[7] S. Checkoway, L. Davi, A. Dmitrienko, A.-R. Sadeghi, H. Shacham, and M. Winandy. Return-oriented programming without returns. In *Proceedings of the 17th ACM conference on Computer and Communications Security (CCS)*, 2010.

[8] S. Checkoway, A. J. Feldman, B. Kantor, J. A. Halderman, E. W. Felten, and H. Shacham. Can dres provide long-lasting security? the case of return-oriented programming and the avc advantage. In *Proceedings of the 2009 Electronic Voting Technology Workshop/Workshop on Trustworthy Elections (EVT/WOTE)*, 2009.

[9] P. Chen, G. Xiao, X. Shen, X. Yin, B. Mao, and L. Xie. Drop: Detecting return-oriented programming malicious code. In *Proceedings of the 5th International Conference on Information Systems Security (ICISS)*, 2009.

[10] M. Christodorescu, S. Jha, S. A. Seshia, D. Song, and R. E. Bryant. Semantics-awaremalware detection. In *Proceedings of the 2005 IEEE Symposium on Security and Privacy*, 2005.

[11] C. Cifuentes and K. J. Gough. Decompilation of binary programs. *Software Practice and Experience, Volume 25 (7): 811-829*, July 1995.

[12] L. Davi, A. Sadeghi, and M. Winandy. Ropdefender: A detection tool to defend against return-oriented programming attacks. In *Proceedings of the 6th ACM Symposium on Information, Computer and Communications Security (ASIA CCS)*, 2011.

[13] A. Dinaburg, P. Royal, M. Sharif, and W. Lee. Ether: Malware analysis via hardware virtualization extensions. In *Proceedings of the 15th ACM conference on Computer and communications security (CCS)*, 2008.

[14] A. Francillon and C. Castelluccia. Code injection attacks on harvard-architecture devices. In *Proceedings of the 15th ACM conference on Computer and Communications Security (CCS)*, 2008.

[15] A. Francillon, D. Perito, and C. Castelluccia. Defending embedded systems against control flow attacks. In *Proceedings of the first ACM workshop on Secure execution of untrusted code (SecuCode)*, 2009.

[16] D. Gao, M. K. Reiter, and D. Song. Binhunt: Automatically finding semantic differences in binary programs. In *Proceedings of the 10th International Conference on Information and Communications Security (ICICS)*, 2008.

[17] S. A. Hofmeyr, S. Forrest, and A. Somayaji. Intrusion detection using sequences of system calls. *Journal of Computer Security, Volume 6(3)*, 1998.

[18] X. Hu, T. cker Chiueh, and K. G. Shin. Large-scale malware indexing using function-call graphs. In *Proceedings of the 16th ACM conference on Computer and communications security (CCS)*, 2009.

[19] R. Hund, T. Holz, and F. C. Freiling. Returnoriented rootkits: Bypassing kernel code integrity protection mechanisms. In *Proceedings of the 18th USENIX Security Symposium*, 2009.

[20] J. O. Kephart and W. C. Arnold. Automatic extraction of computer virus signatures. In *Proceedings of the 4th Virus Bulletin International Conference*, 1994.

[21] T. Kornau. Return oriented programming for the arm architecture. Master's thesis, Ruhr-University Bochum, Germany, 2009. Online: `http://zynamics.com/downloads/kornautim--diplomarbeit--rop.pdf`.

[22] T. Lee and J. Mody. Behavioral classification. In *Proceedings of the 15th Annual EICAR Conference (EICAR)*, 2006.

[23] J. Li, Z. Wang, X. Jiang, M. Grace, and S. Bahram. Defeating return-oriented rootkits with "return-less" kernels. In *Proceedings of the 5th European conference on Computer systems (EuroSys)*, 2010.

[24] K. Onarlioglu, L. Bilge, A. Lanzi, D. Balzarottie, and E. Kirda. G-free: Defeating return-oriented programming through gadget-less binaries. In *Proceedings of The 26th Annual Computer Security Applications Conference (ACSAC)*, 2010.

[25] OpenBSD. W xor X, the openbsd new features. `http://www.openbsd.org/33.html`.

[26] R. Roemer, E. Buchanan, H. Shacham, and S. Savagm. Return-oriented programming: Systems,languages, and applications. *Manuscript*, 2009. Online: `http://cseweb.ucsd.edu/hovav/dist/rop.pdf`.

[27] V. S. Sathyanarayan, P. Kohli, and B. Bruhadeshwar. Signature generation and detection of malware families. In *Proceedings of the 13th Australasian*

conference on Information Security and Privacy (ACISP), 2008.

[28] H. Shacham. The geometry of innocent flesh on the bone: return-into-libc without function calls (on the x86). In *Proceedings of the 14th ACM conference on Computer and Communications Security (CCS)*, 2007.

[29] H. Shacham, M. Page, B. Pfaff, E. Goh, N. Modadugu, and D. Boneh. On the effectiveness of address-space randomization. In *Proceedings of the 11th ACM conference on Computer and Communications Security (CCS)*, 2004.

[30] K. Wang and S. Stolfo. Anomalous payload-based network intrusion detection. In *Proceedings of the 7th International Symposium on (RAID)*, 2004.

[31] C. Willems, T. Holz, and F. Freiling. Toward automated dynamic malware analysis using CWSandbox. *IEEE Security and Privacy, Volume 5(2)*, 2007.

[32] J. Xu, Z. Kalbarczyk, and R. K. Iyer. Transparent runtime randomization for security. In *Proceedings of the 22nd Symposium on Reliable and Distributed Systems (SRDS)*, 2003.

Static Detection of Malicious JavaScript-Bearing PDF Documents

Pavel Laskov
University of Tübingen
Sand 1, 72076
Tübingen, Germany
pavel.laskov@uni-tuebingen.de

Nedim Šrndić
University of Tübingen
Sand 1, 72076
Tübingen, Germany
nedim.srndic@uni-tuebingen.de

ABSTRACT

Despite the recent security improvements in Adobe's PDF viewer, its underlying code base remains vulnerable to novel exploits. A steady flow of rapidly evolving PDF malware observed in the wild substantiates the need for novel protection instruments beyond the classical signature-based scanners. In this contribution we present a technique for detection of JavaScript-bearing malicious PDF documents based on static analysis of extracted JavaScript code. Compared to previous work, mostly based on dynamic analysis, our method incurs an order of magnitude lower run-time overhead and does not require special instrumentation. Due to its efficiency we were able to evaluate it on an extremely large real-life dataset obtained from the VirusTotal malware upload portal. Our method has proved to be effective against both known and unknown malware and suitable for large-scale batch processing.

Categories and Subject Descriptors

D.4.6 [**Software**]: Operating Systems—*Security and Protection*; I.2.6 [**Computing Methodologies**]: Artificial Intelligence—*Learning*

Keywords

Malware detection, malicious JavaScript, PDF documents, machine learning

1. INTRODUCTION

Since the discovery of the first critical vulnerability in Adobe Reader in 2008[1] the Portable Document Format (PDF) has become one of the main attack vectors used by miscreants. PDF-based attacks were the most frequently used remote exploitation technique in 2009 with a proud share of 49%. Two specific PDF-based vulnerabilities were ranked second and fifth among all vulnerabilities discovered in 2009 [1]. Overall, more than 50 vulnerabilities were discovered in Adobe Reader in 2008–2010, which has led to numerous security-related updates. The severity of security problems has somewhat abated with the introduction of the sandboxing technology—which has not been compromised to date—in Adobe Reader X (version 10). The underlying code base, however, still remains vulnerable, and some critical patches were issued earlier this year [2, 3].

The vulnerabilities of Adobe Reader can be classified into three categories. The earliest—and the largest—class of vulnerabilities arises from bugs in the implementation of the Adobe JavaScript API. This API significantly extends the JavaScript functionality in the specific context of PDF documents. The second class of vulnerabilities is rooted in non-JavaScript features of Adobe Reader but typically requires JavaScript for exploitation, e.g. using heap spraying. Examples of such vulnerabilities are the JBIG2 filter (e.g. CVE-2009-0658) and the heap overflow (e.g. CVE-2009-1862) exploits. Finally, the smallest class of vulnerabilities, e.g. the flawed embedded TrueType font handling (CVE-2010-0195), does not involve JavaScript functionality.

Unlike other modern exploitation techniques such as drive-by-downloads, SQL injection or cross-site scripting, the PDF-based attacks have not received significant attention in the research community so far. Previous work in this field has mostly focused on dynamic analysis techniques. For example, well-known sandboxes JSand [7] and CWSandbox [26] have been adapted to the analysis of malicious PDF documents. Due to their heavy instrumentation and security risks associated with dynamic analysis, the practical applicability of such approaches is limited to malware research systems. For the end-user systems, some early work on the detection of potential exploits in PDF documents [13, 21] has gone largely unnoticed, and in practice the detection of malicious PDF documents still hinges upon signatures provided by security vendors.

In this paper, we explore *static* analysis techniques for detection of JavaScript-based PDF exploits. Our aim is to develop efficient detection methods suitable for deployment on end-user systems as well as in the networking infrastructure, e.g. email gateways and HTTP proxies. We present the tool PJScan[2] that is capable of reliably detecting PDF attacks with operational false positive rates in the promille range. The low computational overhead of PJScan makes it very attractive for large-scale analysis of PDF data.

Conceptually, PJScan is closely related to static analysis techniques for detection of browser-based JavaScript attacks. Similarly to the recent work of Rieck et al. [19], our methodology is based on lexical analysis of JavaScript code and uses machine learning to automatically construct models from available data for subsequent classification of new data. The crucial difference from browser-

[1]collab.CollectEmailInfo (CVE-2007-5659): known, as usual, well ahead of an exploit.

[2]The source code of PJScan and its underlying library libPDFJS can be found at `http://sf.net/p/{pjscan|libpdfjs}`.

based JavaScript attacks is that reliable ground truth information is hardly available for PDF documents. It is especially difficult to identify benign JavaScript-bearing PDF documents. First, as our study will show, such examples are indeed much more rare than malicious ones. Second, while it is relatively easy to verify that web content at a certain URL is benign by using Google Safe Browsing[3], it is much more difficult to extract and analyze JavaScript code in PDF documents. These implications necessitate a conceptual redesign of the detection methods. In PJScan, we have to resort to *anomaly detection* to learn only from malicious examples.

Reliable extraction of JavaScript code from PDF documents is itself a major challenge. Not only is PDF very complex, it is also rich with features that can be used for hiding the presence of JavaScript code. It supports compression of arbitrary objects as well as various encodings for the JavaScript content. Such features are routinely used by attackers to avoid detection by signature-based methods. In our experience, none of the previous tools for static analysis of PDF documents, e.g. PDFID[4], JSUNPACK[5], PDF Dissector[6], were able to provide full coverage of the possible locations of JavaScript code in PDF documents. In the preprocessing component of the PJScan, we have developed an interface to a popular PDF rendering library Poppler[7]. Using this interface, our system is able to handle all potential locations of JavaScript known to us from the PDF Reference.

We have evaluated the effectiveness of PJScan on a large real-world dataset comprising 3 months of data uploaded by users to the malware analysis portal VirusTotal[8]. This is the first study of malicious PDF documents carried out at such scale. Our results confirm that there still exist malicious PDF documents that are not recognized by any antivirus system, although the share of novel malicious PDF documents is no longer significant (we have found 52 such documents among more than 40,000 documents classified by VirusTotal as benign). In our experiments, PJScan has attained average detection rates of 85% for known and 71% for previously unknown PDF attacks with the average operational false positive rate of about 0.37%. Due to the difference in the nature of benign data a direct comparison of PJScan with methods for detection of browser-based JavaScript attacks is not possible. Wepawet[9] was the only previous detection method suitable for PDF-based JavaScript attacks. Much to our surprise, while being perfect in terms of false positives and very good in detection of novel PDF attacks (90%), Wepawet has shown poor performance on *known* PDF attacks, for which it only reached the detection accuracy of 63.6%. As a dynamic analysis tool, Wepawet has been conceived for offline analysis and incurs a significant overhead.

1.1 Contributions

In summary, this paper provides the following contributions:

1. **Robust extraction of JavaScript from PDF documents.** We provide a detailed account of the mechanisms for embedding of JavaScript content in PDF documents and present a methodology for reliable extraction of JavaScript code using the open source PDF parser Poppler.

2. **Fully static detection of malicious JavaScript.** We describe a method for discrimination between malicious and benign JavaScript instances based on lexical analysis and anomaly detection. Unlike the previous work, the proposed method does not require manual labeling of data. This is especially important for PDF documents for which it is difficult to verify that a certain document is benign.

3. **High performance.** The key advantage of static analysis is that it allows several orders of magnitude higher processing speed. Our system PJScan has attained the average processing time of less than 50ms per file.

4. **Comprehensive evaluation.** We present the results of a first large-scale evaluation of malicious PDF detection on a real-world dataset comprising more than 65,000 PDF documents. PJScan has detected 85% of known malicious PDF documents compared to *all 42* antivirus scanners deployed by VirusTotal and 71% of previously unknown malicious PDF documents (not detected by any of the VirusTotal's scanners). The promille-range false positive rate of PJScan makes it suitable for practical deployment.

1.2 Paper Organization

The rest of this article is organized as follows. We begin with a brief summary of the main features of PDF and its mechanisms for embedding of JavaScript contents (Section 2). The architecture of PJScan and the methodology used in its specific components is presented in Section 3. In Section 4, we present the data corpus and analyze its statistical features at different representational levels. Our experimental evaluation is presented in Section 5, followed by the discussion of related work in Section 6. Limitations of our methods and potential improvements are discussed in Section 7.

2. PDF AND JAVASCRIPT

Before presenting the technical details of our methods, we briefly summarize the main features of the Portable Document Format and present its syntactic forms used for embedding of JavaScript. A significant portion of the following section contains direct citations from the PDF Reference [15].

2.1 PDF Essentials

A PDF file consists of the following four elements[10]:

- A *header* consisting of the characters `%PDF-` and the version number of the PDF standard used in the file (e.g. 1.1),
- A *body* containing PDF objects with the actual content of the document,
- A *cross-reference table* listing indirectly referenced objects and their location in the file,
- A *trailer*, containing the location of the cross-reference table and some objects in the file body.

The parsing of a PDF file begins with checking the version number and looking at the file trailer for information about the location of the cross-reference table and some special objects in the file body.

[3]Google Safe Browsing API: `http://code.google.com/apis/safebrowsing/`

[4]`http://blog.didierstevens.com/programs/pdf-tools/#pdfid`

[5]`https://code.google.com/p/jsunpack-n/`

[6]`http://www.zynamics.com/dissector.html`

[7]`http://poppler.freedesktop.org/`. Version 0.14.3 was used in our implementation.

[8]VirusTotal, Free Online Malware Scanner, `http://www.virustotal.com/index.html`

[9]Wepawet, `http://wepawet.iseclab.org/index.php`

[10]Many parsers do not strictly follow the PDF Standard. Even the Adobe Reader is notorious for such lack of compliance, e.g. it ignores arbitrary symbols before the header [12] and can dispense with the trailer and cross-references [27].

The PDF standard defines eight basic types of objects:

1. *Boolean* objects take values **true** and **false**.
2. *Integer* and *real* numbers.
3. *Strings* may be stored in two ways:
   - as a sequence of literal characters enclosed in parentheses '(' and ')'.
   - as a sequence of hexadecimal numbers enclosed in angle brackets '<' and '>'.
4. *Names* are sequences of 8-bit characters used as identifiers.
5. *Arrays* are sequences of PDF objects, potentially of different type; arrays can be nested.
6. *Dictionaries* are collections of key-value pairs with keys being names and values being of any PDF object type. Dictionaries are used to describe complex objects such as pages or actions.
7. *Streams* are dictionary objects followed by a sequence of bytes between the words **stream** and **endstream**. Streams can be used to represent large objects, such as images, in a compact way. The content of the byte sequence may be stored in an encoded or compressed form. A special type of streams are *object streams* containing arbitrary PDF objects.
8. The *null* object is denoted by the keyword **null**.

The body of a PDF document is built as a hierarchy of these eight basic types of objects linked together in a semantically meaningful way to describe pages, multimedia, outlines, annotations, etc. A central role in the hierarchy belongs to the *Catalog* dictionary pointed to by the `/Root` entry of the cross-reference table. It serves as the root of a tree-like structure describing the document content.

Objects can be assigned a unique identifier consisting of an object number and a generation number (a sort of a version number). Objects that have a unique identifier can be referenced from other objects using an *indirect reference* written as a sequence of the object number, the generation number and the capital letter 'R'. For example, `23 0 R` refers to an object with the object number 23 and the generation number 0. PDF allows encryption of the contents of strings and streams.

2.2 JavaScript in PDF

PDF provides several mechanisms for inclusion of JavaScript code. These mechanisms are important for the realization of interactive features, such as forms, dynamic content or 3D rendering. Some PDF usage scenarios relying on these features cannot be realized without JavaScript.

The main indicator for JavaScript code is the presence of the keyword `/JS` in some dictionary. The JavaScript source is supplied directly as one of the two possible string types (literal or hexadecimal) or stored in another object pointed to by an indirect reference. In the latter case, it is usually stored in a compressed or encrypted form in a stream attached to that object. Examples of typical syntax for embedding of JavaScript code are shown in Fig. 1.

A simple search for `/JS` patterns in PDF files – as it was realized in some tools for the analysis of PDF documents, e.g. PDFID – does not suffice for identification of JavaScript locations. It can be easily evaded by placing objects containing dictionaries with the keyword `/JS` into object streams. Due to stream compression the keyword `/JS` is not visible in plain text. The simple search may also

```
1 0 obj <<                1 0 obj <<
  /Type /Catalog            /Type /Catalog
  /Pages 2 0 R              /Pages 2 0 R
  /OpenAction <<            /OpenAction <<
    /S /Rendition             /S /JavaScript
    /JS 23 0 R                /JS (alert('Hello World!');)
  >>                        >>
>>                        >>
endobj                    endobj
```

Figure 1: Exemplary syntactic constructs for embedding of JavaScript in PDF documents. Left: code is placed in another object pointed to by an indirect reference (not shown). Right: code is supplied as a literal string.

yield multiple references to identical code if different revisions of the same content are present.

In order to reliably extract JavaScript code, the documents must be processed at the semantic level, i.e. considering potential uses of JavaScript in the context of other objects in a document. In general, the use of JavaScript code in PDF documents is bound to the so-called *action dictionaries*. Such dictionaries may be tagged by a keyword/value pair `/Type/Action`, but unfortunately such explicit qualification is optional and cannot be relied upon. A mandatory feature of all action dictionaries is the keyword `/S` which may take on 18 different name values. Two of such values, `/JavaScript` and `/Rendition`, are important for the search for JavaScript code. The former must, and the latter may have a keyword `/JS` [15], as shown in Fig. 1. The content associated with the keyword `/JS` must use the PDFDocEncoding (as defined in [15]) or the UTF-16BE (big-endian) Unicode encoding. In the rest of this article we denote JavaScript source code located in or referred to by one *JavaScript* or *Rendition* action dictionary as a *JavaScript entity*.

JavaScript or *Rendition* action dictionaries can be found at the following locations of the PDF object hierarchy:

- The Catalog dictionary's `/AA` entry may define an additional action specified by a JavaScript action dictionary.
- The Catalog dictionary's `/OpenAction` entry may define an action to be run when the document is opened.
- The document's name tree may contain an entry 'JavaScript' that maps name strings to document-level JavaScript action dictionaries executed when the document is opened.
- The document's *Outline* hierarchy, referenced by means of the 'Outlines' entry of the Catalog dictionary, may contain references to JavaScript action dictionaries.
- Pages, file attachments and forms may also contain references to JavaScript action dictionaries.

Besides being directly embedded in a PDF file, JavaScript code may also reside in a different file on a local machine or even be retrieved from a remote location using the directives `/URI` or `/GoTo`. JavaScript also supports dynamic code execution using the `eval()` function or its equivalent, `setTimeOut()`. Such mechanisms are difficult for static analysis; however, they are launched from an existing *entry point* code inside a document.

3. SYSTEM ARCHITECTURE

The architecture of our PDF scanner PJScan is shown in Fig. 2. Conceptually, our system consists of the feature extraction and the learning components. The feature extraction component searches

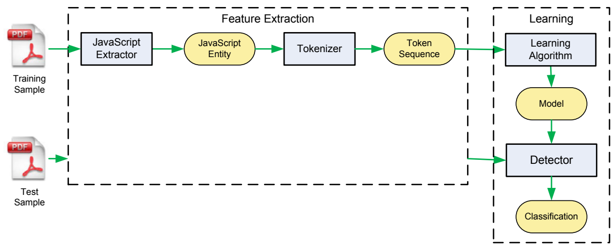


Figure 2: Architecture of PJScan

for JavaScript code embedded in a document and performs lexical analysis on it. The resulting token sequence is used as an input to the learning component. The learning component is first trained on examples of *malicious* documents. It produces a model for the JavaScript content in a malicious PDF document. Classification of new documents is performed using this model. A detector measures the deviation of a new document from a learned model and compares it against a predefined threshold (the threshold can also be automatically determined at the training stage). Documents that are close to a learned model are classified as malicious and otherwise as benign. The functionality of specific components depicted in Fig. 2 is described below.

3.1 Extraction of JavaScript Content

The main challenge of the extraction of JavaScript content lies in the decoding of object streams and the handling of the encoding used for JavaScript content. Furthermore, a parser must be robust against potential incompatibilities with the PDF Standard. For this reason, contrary to the approach taken in [24], we have decided against the parsing of PDF files "by hand" and tailored a popular open source PDF parser Poppler to the needs of our analysis.

Our JavaScript extractor begins with opening the PDF file and initializing Poppler and its internal data structures. Next, the Catalog dictionary is retrieved which serves as the starting point in the search for action dictionaries. All candidate locations listed in Section 2.2 are checked, and the found action dictionaries are queried for their type. If the type is *Rendition* or *JavaScript* and a dictionary contains the /JS key, the value of this key (or the referenced object in case of an indirect reference) is retrieved. The JavaScript entity is then decompressed and decoded if necessary.

The peculiarity of our approach is that we fully process only those objects in which *JavaScript* and *Rendition* action dictionaries can potentially occur. This strongly reduces the computational effort for extraction of JavaScript content and is crucial for batch processing of large datasets. Files that do not contain any JavaScript are not processed beyond the extraction stage.

3.2 Lexical Analysis

Two factors motivate the use of lexical analysis for the detection of malicious JavaScript code. First, we believe that at the text level, accurate discrimination between malicious and benign programs is not possible. Second, malicious JavaScript code is usually – sometimes insanely – obfuscated. We have also observed obfuscation in benign JavaScript entities extracted from PDF documents. Hence we have decided to use an intermediate representation – the set of lexical tokens – to capture the salient properties of code in subsequent analysis.

The lexical analysis can be efficiently carried out by the state-of-the-art open source JavaScript interpreter SpiderMonkey[11] developed by the Mozilla Foundation. To use it as a token extractor, we have patched SpiderMonkey to stop short of byte-code generation. Our extractor queries SpiderMonkey for tokens until an end-of-file or an error is encountered. Tokens representing various syntactic elements of the JavaScript language, e.g. identifiers, operators, etc., are represented as symbolic names with integer values ranging from -1 (TOK_ERR) to 85.

Some semantics of the code is lost during lexical analysis. For example, all identifiers get assigned the same token TOK_NAME (regardless of their names), calls to different functions with identical signatures are translated into the same token sequences, and so on. As a result, JavaScript entities that are distinct at the source code level may be non-distinct at the token sequence level.

The following example illustrates the tokenization process. The malicious JavaScript entity

```
bvb('var lBvXSUfYYL7RK = ev' + 'al;'); // a real example
lBvXSUfYYL7RK('var uzWPsX8 = this.info' +
   z("%2e%46%61%6b") + 'erss;');
```

is transformed into the following sequence of tokens, shown in their order from top to bottom:

Value	Symbolic name	Description
29	TOK_NAME	identifier
27	TOK_LP	left parenthesis
31	TOK_STRING	string constant
15	TOK_PLUS	plus
31	TOK_STRING	string constant
28	TOK_RP	right parenthesis
2	TOK_SEMI	semicolon
29	TOK_NAME	identifier
27	TOK_LP	left parenthesis
31	TOK_STRING	string constant
15	TOK_PLUS	plus
29	TOK_NAME	identifier
27	TOK_LP	left parenthesis
31	TOK_STRING	string constant
28	TOK_RP	right parenthesis
15	TOK_PLUS	plus
31	TOK_STRING	string constant
28	TOK_RP	right parenthesis
2	TOK_SEMI	semicolon
0	TOK_EOF	end of file

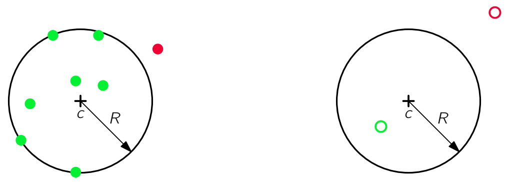
Besides the tokens recognized by SpiderMonkey, we have defined extra tokens that are indicative of malicious JavaScript entities. The newly-introduced tokens are listed in the following table. The impact of these tokens on the classification performance of PJScan is evaluated in Section 5.4.

Value	Symbolic name	Description
101	TOK_STR_10	a string literal of length < 10
102	TOK_STR_100	a string literal of length < 100
103	TOK_STR_1000	a string literal of length < 1,000
104	TOK_STR_10000	a string literal of length < 10,000
105	TOK_STR_UNBOUND	a string literal of length > 10,000
120	TOK_UNESCAPE	a call to unescape()
121	TOK_SETTIMEOUT	a call to setTimeOut()[12]
122	TOK_FROMCHARCODE	a call to fromCharCode()[13]
123	TOK_EVAL	a call to eval()

[11]http://www.mozilla.org/js/spidermonkey/

[12]In PDF, the function setTimeOut() of the app object can be used as a replacement for eval() to execute arbitrary JavaScript code after the specified timeout.

[13]fromCharCode() is a static method of the *String* object that converts Unicode values to characters. In malicious documents, it is used to decode encoded strings for execution using eval().



(a) Learning stage: the center c and the radius R of the sphere are determined.

(b) Classification stage: new data are accepted (green) or rejected (red).

Figure 3: OCSVM operation

3.3 Learning and Classification

In the last step in our processing chain, the learning component of PJScan determines whether a PDF file is benign or malicious. Prior to deployment, it must be trained on a representative set of malicious PDF files. The training results in a model of malicious JavaScript entities in a PDF document. At the deployment stage, classification of new PDF documents is carried out with the help of the learned model. After the feature extraction steps described in Sections 3.1 and 3.2 are completed, the set of tokens from a new document is tested for proximity to the model.

In our system, we use the One-Class Support Vector Machine (OCSVM) [23] as the learning method[14]. Its main advantage is that it only needs examples of one class to build a model. This is necessary since examples of benign PDF documents with JavaScript content are quite rare, and it takes a lot of manual effort to verify that they are benign. On the other hand, examples of malicious PDF documents abound on malware collection systems, and their maliciousness can be ascertained with high confidence if they are detected by antivirus systems.

The learning stage of OCSVM (cf. Fig. 3(a)) amounts to finding the center c and the radius R of a high-dimensional hypersphere such that the total percentage of all data points lying outside of the hypersphere is at most ν. A hypersphere may be extended to arbitrary surfaces by a non-linear transformation to a special feature space equipped with the so-called "kernel function". The kernel function type and the training rejection rate ν are the only parameters to be specified for training of OCSVM. The learned model comprises the center of the sphere c and the radius R.

The classification stage of OCSVM involves the calculation of the distance between the data point to be classified and the center of the hypersphere. If the distance is greater than R (the data point lies outside of the hypersphere), then it is considered an anomaly and is treated as benign. The radius thus serves as a threshold that is automatically determined at the training stage. The classification stage of OCSVM is illustrated in Fig. 3(b).

OCSVM cannot be directly applied to token sequences emitted by PJScan's feature extraction component. The reason for this is that OCSVM expects the data points to be numeric values lying in a high-dimensional space equipped with typical mathematical operations such as addition, multiplication with a constant and an inner product. Sequential data does not form such a space: it is not immediately clear how to add or multiply two strings. A solution to this problem involves a well-established technique of embedding sequences in metric spaces [20]. By counting the occurrences of substrings in data points and assigning the resulting numeric values to coordinate axes, the required mathematical properties can be enforced.

[14]The popular open source SVM implementation LibSVM [6], version 2.86, patched to support one-class SVM, was used in our experiments.

The embedding of sequences provides an elegant way for handling multiple JavaScript entities in the same file. To obtain an aggregated representation of all JavaScript entities, it suffices to add them using the addition operation provided by the embedding. To avoid the dependence on sequence length, the values in individual dimensions are binarized (by setting any positive values to 1) and normalized so that the Euclidean norm of the resulting vectors is equal to 1.

4. DATA COLLECTION AND ANALYSIS

The success of any learning-based approach crucially depends on the quality of data available for training. Likewise, the viability of a learned model can only be demonstrated on up-to-date real data. The evaluation of our method rests on an extensive dataset collected from the research interface to the popular malicious software portal VirusTotal. VirusTotal is a web service that enables ordinary users to upload suspicious files for a scan by 42 antivirus engines. Our dataset comprises 65,942 PDF documents with the total size of nearly 59GB. This data has revealed some interesting features, and is worth looking at in some detail.

We downloaded three batches of data on November 3, 2010, January 19, 2011 and February 17, 2011 each containing all PDF files available on VirusTotal at a given time. The data is kept only for 30 days, and there is surprisingly little overlap between subsequent months[15]. We have observed at most 200 identical files across different snapshots. We have split our corpora into two parts, the "detected" sub-corpus in which documents were flagged as malicious by at least one scanner, and the "undetected" sub-corpus containing supposedly benign data.

It is instructive to look at the statistical properties of our data presented in Table 1. One can notice interesting effects in the collected data. The average file size in the "detected" corpora (0.106MB) is about 13 times smaller than in the "undetected" ones (1.390MB). This shows that malicious PDF files do not contain a lot of meaningful content, which is confirmed by a manual investigation of some of these files. The percentage of files with JavaScript in the "detected" corpora (59.5%) is about 25 times higher than in the "undetected" corpora (2.4%). This is a strong indicator that JavaScript plays a crucial role in PDF-related exploits.

Considering only the files containing JavaScript one can see that the average number of JavaScript entities per file in the "detected" datasets (7.2) is around 33 times *smaller* than in the "undetected" datasets (241.1). This observation seems counter-intuitive but it turns out that "undetected" data usually contains hundreds of very simple JavaScript entities like `this.zoom=100;this.pagenum=39`. Similarly, distinctness of JavaScript entities at the code level is 3.2 times higher in "detected" corpora than in "undetected" (16.9% vs. 5.2%). These findings suggest that non-malicious usage of JavaScript in PDF documents essentially boils down to boring and redundant code!

Similar effects take place at the token level (the second row from the bottom in Table 1). One can observe a further decrease of distinctness (6,419 vs. 35,990, or 17.8%) due to lexical analysis. The "detected" sub-corpora are 7.5 times more distinct than the "undetected" sub-corpora. Finally, the last row in Table 1 reveals that many files contain identical sets of token sequences, which can be explained by common code reuse in both types of files.

To enable the quantitative evaluation of detection accuracy in the forthcoming experiments, we have manually labeled the "undetected" part of our data. Among 960 benign files with JavaScript,

[15]In fact, we were originally unaware of the 30 day lifespan and started a periodic collection of snapshots only in January.

	03. Nov. 2010		19. Jan. 2011		17. Feb. 2011		Total
	detected	undet.	detected	undet.	detected	undet.	
Dataset size	873MB	13GB	429MB	13GB	1.5GB	29GB	59GB
Files in the dataset	7,592	7,768	6,465	9,993	11,634	22,490	65,942
Files containing JavaScript	6,626	272	1,127	196	7,526	492	16,239
JavaScript entities	26,372	75,199	33,418	42,265	50,269	113,994	341,517
Distinct JavaScript entities	8,597	5,178	2,376	3,774	9,238	6,827	35,990
Distinct token sequences	1,108	429	815	356	2,947	764	N/A
Distinct files on the token sequence level	538	115	358	95	1,900	237	N/A

Table 1: Statistics of PDF documents collected from VirusTotal

we found 52 PDF documents that we believe to have been falsely classified as benign by all antivirus engines at VirusTotal. No cases were found where PDF files belonging to the same group of distinct files at the token level were assigned different labels.

5. EXPERIMENTAL EVALUATION

The real-world nature and the sheer size of the VirusTotal data make our evaluation especially challenging. First, the distinction between "detected" and "undetected" corpora is somewhat vague, as classifications by antivirus engines cannot be fully trusted. Second, the huge size of the "detected" corpus makes its manual analysis infeasible. On the other hand, the small size of the labeled JavaScript-bearing part of the "undetected" corpus is too small to be used for training purposes.

As the baseline for comparison we consider Wepawet, a web-based service based on JSand [7]. Wepawet performs both static and dynamic analysis of PDF files based on their JavaScript content and can detect malware that it has a signature for (labeled as *malicious*), as well as unknown malware (labeled as *suspicious*) using statistical features. In the evaluation, we treat both categories as detections. Similar to our system, Wepawet generally does not recognize PDF malware that does not use JavaScript. Table 2 shows Wepawet's classification on the "detected" and "undetected" parts of all three corpora at our disposal. In some cases file uploads were rejected by Wepawet, referred to as *fail*, or resulted in internal errors despite multiple submissions, referred to as *error*. We treat such cases (about 1.7% of the total data) as benign.

	03. Nov. 2010		19. Jan. 2011		17. Feb. 2011	
	det.	undet.	det.	undet.	det.	undet.
Fail	12	38	9	25	19	73
Error	15	1	5	0	83	0
Benign	3,860	212	502	167	1,050	397
Suspicious	1,474	11	149	0	257	0
Malicious	1,265	10	462	4	6,117	22

Table 2: Wepawet classification results

5.1 Objectives and Evaluation Criteria

Our experiments address the following questions:

1. How well do PJScan and Wepawet detect known malicious documents? This question may appear meaningless: why bother detecting something that is already detected? In practice, however, it is impossible to deploy all 42 antivirus engines from VirusTotal. For a single method, attaining the detection accuracy close to that of 42 established antivirus products is still a very challenging goal[16]. The corresponding quality measure is the *true positive rate on known attacks* TP_N defined as the ratio of the number of files in the "detected" corpus classified as malicious to the total number of JavaScript-bearing files in that corpus.

2. How well do both methods detect attacks that were missed by all 42 VirusTotal engines? We consider documents in the "undetected" corpus as novel attacks if they are classified as malicious during manual analysis. The *true positive rate on unknown attacks* TP_U is defined as the ratio of the number of files in the "undetected" corpus classified as malicious to the total number of malicious JavaScript-bearing files in that corpus.

3. How many normal documents are classified as malicious by the methods in questions? The *laboratory false positive rate* FP_L is defined as the ratio of the number of files in the "undetected" corpus classified as malicious to the total number of benign JavaScript-bearing files in the "undetected" corpus. The *operational false positive rate* FP_{OP} is the ratio of the number of files in the "undetected" corpus classified as malicious to the total number of benign files in the "undetected" corpus.

The distinction between the laboratory and the operational false positive rates is essential for estimation of the expected impact of false positives in practical deployment.

5.2 Experimental Protocol

Our experiments were carried out using the following procedure. We merged all 3 corpora from different dates keeping only the distinction between "detected" and "undetected" parts. We then randomly split the full "detected" corpus in two non-overlapping parts such that the corresponding sets of token sequences are of the same size. Due to a significant redundancy of token sequences this results in two sets of files that are different in size. One of these half-corpora is used to train PJScan, the other half is used to evaluate TP_K. To decrease the impact of non-determinism via random splitting, we repeat the experiment the second time by swapping the training and the evaluation datasets and averaging the detection accuracy. This process is known as *2-fold cross-validation*.

To determine the detection accuracy on unknown data, we apply the trained model on the full "undetected" corpus. We use the ground truth information to compute TP_U, FP_L and FP_{OP}. The reported results are also averaged over the two partitions of the training data.

[16]Unfortunately we cannot compare any method against the *best* detector at VirusTotal's. The labels in batch data from VirusTotal reflect only the number of detections but not the specific engines that classified a document as malicious.

Since the models used in Wepawet do not depend on our training data (but rather on the data its statistical part was trained on), the results presented for this method reflect the accuracy of scanning a complete respective dataset ("detected" or "undetected").

Some preliminary experimentation was needed to choose the parameters of OCSVM used in our method. We chose the training rejection rate $\nu = 0.15$ and the n-gram length of 4, which seem to provide the best trade-off between the true positive and false positive rates. The full results of our preliminary screening for optimal parameters cannot be presented due to space constraints.

5.3 Experimental Results

The results of a comparative evaluation of PJScan and Wepawet according to the criteria specified in Section 5.1 are presented in Table 3. Two configurations of PJScan were considered: using only native JavaScript tokens and using a set of additional heuristic tokens introduced in Section 3.2. It can be seen that PJScan significantly outperforms Wepawet on the known malicious data but performs less accurately on previously unknown attacks. Most of failed detections were caused by 11 files which are redundant at the token level and contain the following code[17]:

```
app.setTimeOut(this.info.dgu,1)
```

In this example, the attack code resides *not in a JavaScript entity* but in the *Info* dictionary[18]. It can be still accessed by a very short entry-point JavaScript code above as text and gets interpreted as JavaScript by calling the function `setTimeOut()` which is equivalent to `eval()`. With an exception of this kind of attack, the detection rate of PJScan would have also reached the 90% mark.

It is not clear to us why Wepawet has performed relatively poorly on known malicious data. In a related comparative evaluation against Cujo [19] in the context of web-based JavaScript attacks (drive-by-downloads), Wepawet was a clear winner with a detection rate of 99.8% compared to 94.4%. Most likely, the reason for worse performance of Wepawet in our experiments lies in technical problems with the extraction of JavaScript code from PDF documents.

Detection method	TP_K	TP_U	FP_L	FP_{OP}
PJScan (native tokens only)	84.80	71.15	16.35	0.3694
PJScan (with extra tokens)	85.17	71.15	17.35	0.3918
Wepawet	63.60	90.38	0.0	0.0

Table 3: Detection performance overview

A relative disadvantage of PJScan is the high false-positive rate. Measured against only the JavaScript-bearing benign documents it reaches the painful 16-17%; however, due to the rare presence of JavaScript code in benign documents, its operational false-positive rate remains acceptable and corresponds, for our data, to 1.7 false alarms per day.

One can also see that heuristic tokens do not improve the performance of PJScan and even lead to a slight degradation of the false-positive rate. The causes for this effect as well as for the false positives are elucidated in the following section.

5.4 Significant Features

As noted by Sommer and Paxson [22], a security practitioner would always be interested to know what a learning method has actually learned. The model created by the OCSVM (the center c of the sphere) produces a numeric ranking of essential features encountered in malicious JavaScript code. Since no benign data is used for training, this ranking does not reflect the differences between two classes but rather describes only one class known to it. Examples of the 5 most important and the 5 least important features in one of the models learned by PJScan (created for one half of the data) are shown in Table 4.

Although these features do not look particularly malicious, the top 5 features clearly correspond to typical lexical patterns of programming languages: member function dereferencing (Feature 1), string variable assignment (Feature 2), function calls (Features 3 and 4) and variable declarations (Feature 5). On the other end of the spectrum are the features that are obviously very atypical for programming languages.

The scoring of a new data point in the detection phase involves the identification of an overlapping subset of features between this data point and the learned model. The smaller the "weighted overlap" between the new point and the center (i.e. the sum of the weights in the model corresponding to the common features), the larger the distance from the center. This property is confirmed by the examples of accepted and rejected points presented below.

For the accepted points (Table 5, one true positive and one false positive), the main contributions are made by the top features of the trained model. Such points are virtually indistinguishable in our model, and this explains a high "laboratory" false positive rate observed in our experiments. It turns out, however, that *very few benign examples* share the "normal" programming language features captured by the learned model. For the two examples of rejected points (Table 6, one true negative and one false negative) the top features have much lower ranks in the learned models. The majority of benign examples have a small "weighted overlap" with the model and hence are rejected.

The investigation of significant features in our models suggests that the key property that enables effective discrimination between malicious and benign code in PDF documents is the fact that benign usage of JavaScript is very rudimentary from the programming point of view. Anecdotally, the benign example with the highest rejection score corresponds to the code `print(true)`.

5.5 Throughput

The throughput of PJScan was tested on a commodity PC with a quad-core Intel Core i7 860 CPU, 8 GB of RAM and a 7,200rpm SATA hard disk drive. Eight processes were run concurrently for performance measurement.

Each phase of PJScan was run on a respective data partition (training on one half of "detected" corpus, evaluation on the other half and on the full "undetected" corpus). Unlike the accuracy measurement, we learned and classified using *all* files ignoring their redundancy. Learning with thousands of files instead of a few hundred distinct token sequences reduces performance, but due to the fast learning and classification algorithms the difference is negligible. Processing times for all stages of our method are shown in Table 7. In total, parsing of 65,942 PDF files, tokenization of 341,517 JavaScript entities, learning on 15,279 "detected" files with JavaScript and classification of 960 "undetected" files with JavaScript took 1,547 seconds (about 25 minutes). All measurements are expressed in wall clock time[19].

[17]All examples differ in the name for the member of the `this.info` dictionary (in this case, `dgu`).

[18]An Info dictionary is used to store meta-data about the PDF file, such as author name, the software used to create it, etc.

[19]Wall clock time measures real time that elapses between the beginning and the end of a task. It includes CPU time, I/O time and any overhead such as the time process spends waiting for execution. It is a good indicator of real performance but is affected by system load.

Top 5			Bottom 5		
Rank	Weight	Feature	Rank	Weight	Feature
1	0.05285	`NAME . NAME (`	4051	2.285e-05	`) NAME ( THIS`
2	0.05106	`NAME ASSIGN STR ;`	4052	2.285e-05	`+- NAME !== NAME`
3	0.05092	`NAME ( NAME )`	4053	2.285e-05	`] ) - NAME`
4	0.04574	`( NAME ) ;`	4054	2.285e-05	`NAME ] ) -`
5	0.04314	`; VAR NAME ASSIGN`	4055	1.865e-05	`TRUE } ; IF`

Table 4: Features of the center point

True positive top 5			False positive top 5		
Rank	Weight	Feature	Rank	Weight	Feature
1	0.00456	`NAME . NAME (`	1	0.00554	`NAME . NAME (`
2	0.00441	`NAME ASSIGN STR ;`	2	0.00535	`NAME ASSIGN STR ;`
3	0.00439	`NAME ( NAME )`	5	0.00452	`; VAR NAME ASSIGN`
4	0.00395	`( NAME ) ;`	6	0.00413	`; NAME ( NAME`
5	0.00372	`; VAR NAME ASSIGN`	7	0.00386	`NAME ( STR )`

Table 5: Features of TPs and FPs

True negative top 5			False negative top 5		
Rank	Weight	Feature	Rank	Weight	Feature
7	0.00390	`NAME ( STR )`	1	0.01593	`NAME . NAME (`
8	0.00390	`VAR NAME ASSIGN NAME`	98	0.00490	`. NAME . NAME`
10	0.00359	`) ; NAME ASSIGN`	141	0.00394	`( THIS . NAME`
14	0.00338	`THIS . NAME (`	154	0.00372	`THIS . NAME .`
15	0.00338	`ASSIGN NAME . NAME`	355	0.00177	`NAME ( THIS .`

Table 6: Features of TNs and FNs

One can see that JavaScript extraction is the most time-consuming part of PJScan. It has been observed that this operation takes only 2,041 seconds using a single process, with an average processing time of 31 milliseconds per file. The overall CPU usage was very low (up to 40%, with I/O waiting of up to 30%), while at the same time disk utilization remained above 95% during the extraction phase. One can conclude that disk throughput represents the main performance bottleneck for this application. Using a faster storage device or reading files through a fast network is likely to improve the performance of PJScan.

The throughput calculation for different stages of PJScan is presented in Table 8. It shows that the throughput varies strongly between the "detected" and "undetected" files since they vary in file size and the number and size of JavaScript entities. The average throughput of 303.5Mbps is suitable for batch processing tasks even for organizations that have a very high volume of PDF traffic. The average processing time per file is 23 milliseconds. To the best of our knowledge, no other software package achieves lower processing times.

Time	Extractor	Tokenizer	Learner	Classifier
Total	1,356s	180s	10.19	0.014s
Average	0.0205s	0.0032s	N/A	0.000015s
Std. dev.	0.0015s	0.0392s	N/A	0.000009s
Percentage	87.65%	11.63%	0.66%	0.0009%

Table 7: Processing time for different stages of PJScan in batch execution mode

	Detected	Undetected	All files
Total time	339s	1208s	1547s
Average file size	0.106MB	1.39MB	0.89MB
Files per second	75.8	33.3	42.6
Data throughput	64.3Mbps	370.5Mbps	303.5Mbps
Seconds per file	0.013s	0.030s	0.023s

Table 8: Throughput characteristics of PJScan

6. RELATED WORK

As it was already mentioned in the introduction, detection of malware in PDF documents has not been extensively studied in the research literature before. The early approaches to identification of malware in PDF documents [13, 21] were based on *n*-gram analysis of *raw document content*. The scope of experimental evaluation in this work was rather limited. It included self-generated malicious PDF documents as well as a relatively small number of examples (less than 300) from the outdated VXHeavens malware repository. Due to the wide-spread use of evasion techniques in modern PDF malware, especially object compression and code-level obfuscation, we believe that the analysis of raw content of PDF documents is no longer adequate. Hence the approach taken in PJScan is fundamentally different from the above mentioned work in that our methods spend a lot of effort in discovering and utilizing the appropriate lexical features of PDF.

The recent work on analysis of PDF documents has emerged from existing tools for static and dynamic analysis. Besides the malware analysis portal Wepawet considered in our experiments, some other tools use a combination of static and dynamic analysis tools. MalOffice [10] uses static and dynamic techniques as well as some heuristics. Its static analysis is based on the utility pdftk[20], and the dynamic analysis builds on CWSandbox [26]. Detections are made by combining scores from various heuristics and policies attached to the analysis tools. Another combination of static and dynamic analysis was used in MDscan recently proposed in [24]. From the architectural point of view, MDscan is similar to our approach. It also uses static analysis to extract JavaScript content (using a self-made parser) and a heuristic approach for the extraction of JavaScript code. The extracted code is interpreted using SpiderMonkey. Detection is carried out at the dynamic stage by using the shellcode detection tool Nemu [16]. The method was evaluated on a set of 197 malicious PDF documents artificially generated using the Metasploit framework and 2000 benign documents. Compared to MDscan, we only use SpiderMonkey for token extraction and perform detection statically, which brings a performance improvement of two orders of magnitude.

A significant body of prior work has addressed the detection of malicious JavaScript in web content, especially in the context of drive-by-downloads. One cannot directly compare the accuracy of such methods with PJScan due to the fact that the data corpora used for the experimental evaluation of respective methods are very different. We will hence focus on methodical comparison of our approach with such methods.

Similar to PDF malware, the methods for detection of malicious JavaScript in web content can be classified into static, dynamic and hybrid. Purely dynamic methods deploy various techniques for monitoring the run-time execution of processes accessing web content, e.g.: full-fledged host virtualization [25], client virtualization [14], instrumentation of a JavaScript engine [11] or heap monitoring [18]. Dynamic methods have high detection accuracy and are hardly prone to false positives. Due to their performance overhead they are usually limited to "post-mortem" analysis.

Hybrid methods aim to minimize run-time overhead while retaining high detection accuracy. Several such methods have methodical affinity with PJScan. JSand [7] uses instrumented versions of the HtmlUnit[21], a Java-based browser simulator, and the Mozilla's Rhino[22] interpreter to extract heuristic features while monitoring the execution of JavaScript code. These features are used to train an anomaly detection system by running JSand on benign web pages. Cujo [19] is another interesting combination of static and dynamic methods. Its static part is similar to PJScan (with the exception of anomaly detection instead of two-class classification in its learning component); its dynamic component extracts symbolic features from a light-weight sandbox ADSandbox [9] and deploys similar *n*-gram analysis and learning techniques as the static part. A "mostly static" detection system Zozzle [8] attempts to avoid dynamic analysis but still needs it to unravel source-code obfuscation before using statistical feature extraction and supervised learning for the classification part. Compared to these hybrid methods, PJScan uses "reverse" anomaly detection—since only malicious data is widely available for PDF documents—and completely dispenses with run-time analysis. Another hybrid method has been proposed by Provos et al. [17]; however, the lack of a technical presentation in this reference prevents us from a detailed comparison.

The only method that can be classified as fully static is Prophiler [5] which deploys techniques similar to JSand except that its features are extracted from a JavaScript engine at the parsing stage *without running the code*. (A similar idea is used in our method but one step earlier, by stopping SpiderMonkey after the lexical analysis.) However, Prophiler has a high false positive rate and is intended to be used as a filter for a subsequent dynamic analysis.

[20]`http://www.pdflabs.com/tools/pdftk-the-pdf-toolkit/`.

[21]`http://htmlunit.sourceforge.net/`

[22]`http://www.mozilla.org/rhino/`

7. DISCUSSION AND LIMITATIONS

The reported experimental results confirm the practical feasibility of the static, learning-based approach for detection of malicious JavaScript-bearing PDF documents. The preprocessing component of PJScan can be very helpful for a security administrator to manually extract and analyze JavaScript code in PDF documents. The main benefit of the learning component of PJScan is the ability to *extract knowledge from large-scale malware corpora*. PJScan enables one to derive light-weight models from heuristic knowledge of several dozen antivirus engines and tens of gigabytes of collected data. Such models can be deployed with no manual interaction and negligible performance overhead (<50ms per file). The operational false-positive rate of less than 0.4% is admissible in practice; even for a highly visible site like VirusTotal with a strong bias for suspicious data, this corresponds to an average rate of 1.7 false alarms per day (148 out of ca. 40,000 benign documents over 90 days).

The high "laboratory" false-positive rate of PJScan (i.e. the rate measured only for those benign files that contain JavaScript) indicates that our current learning setup may indeed have difficulty with accurate discrimination between malicious and benign JavaScript content. This observation is also indirectly supported by our analysis of the learned features. Learning from two classes, as it has been done in the related work on web-based JavaScript content, e.g. [19, 5, 8], may be the right way to avoid this limitation. However, benign JavaScript-bearing PDF data is currently not available in sufficient quantity to evaluate this scenario for PDF documents.

Another limitation of the current version of PJScan is its susceptibility to certain kinds of obfuscation. An exemplary obfuscation technique that is difficult for our method is the use of short JavaScript entry-point code which fetches further code from document locations where JavaScript code cannot be expected (cf. Section 5.3). There are two potential ways to address this limitation. One can use the "mostly static" technique proposed in Zozzle [8] in which compilation requests to a JavaScript engine are intercepted to obtain all code sent for execution. While this technique offers a guaranteed access to unobfuscated code, it may be hampered by just-in-time compilation used in JavaScript engines and eventually produce highly fragmented code. It should be also noted that this idea would be difficult to implement for PDF-based JavaScript code since Adobe provides an extensive PDF-specific API. Another way of dealing with obfuscation is to collect syntactic information from a parser and use compiler optimization to factor out obfuscations.

An attacker may also attempt to use the fact that the models used for detection are derived from data. A taxonomy of attacks against learning algorithms has been recently proposed by Barreno et al. [4]. Following this taxonomy, we remark that causative attacks, i.e. attacks against the training data, do not constitute a serious threat to our approach. We use data from an established malware repository and assume that integrity of this repository cannot be compromised. Even if an attacker submits own data to this repository, he will know how this data is classified by antivirus engines but cannot influence this classification. More realistic are attacks from the exploratory category, i.e. attacks staged at the detection stage. One potential attack strategy is to insert some useless code to make a new JavaScript entity look "anomalous". This attack may indeed be quite potent if an attacker knows the true profile of "normal" malicious data. Since he neither has access to nor can manipulate the training data, we believe that in practice guessing what kind of useless code should be added can be a difficult task.

8. CONCLUSIONS

We have proposed a new static approach to detection of malicious JavaScript-bearing PDF documents. The main advantages of our approach are its high performance and no need for special instrumentation, such as virtual machines or sandboxing. It can attain about 85% of the detection accuracy of *all* antivirus engines at VirusTotal with the performance overhead of less than 50ms per file. It is only marginally affected by text-level obfuscation since the resulting JavaScript code remains very conspicuous at the lexical level. Due to these advantages our method can be used as a standalone application on end-user systems or even be integrated as a filtering tool in email gateways and HTTP proxies.

The computational efficiency of our system PJScan has enabled us to evaluate it on an *unprecedentedly large* real-life data corpus (over 65,000 PDF documents) collected from VirusTotal. This evaluation has confirmed a high detection accuracy of our method for both known and unknown malware. PJScan is more prone to false positives than state-of-the-art dynamic approaches; however, its operational false positive rate still lies in the promille range, which is feasible for practical deployment.

Our future work will address a potential interaction of static and dynamic analysis techniques in order to unravel code-level obfuscation typical for JavaScript attacks. We anticipate that some degree of dynamic analysis can be carried out prior to actual code execution without a significant performance overhead. We also intend to investigate more extensive static analysis techniques, such as syntactic analysis and compiler optimization, to obtain features that better reflect the true semantics of the JavaScript code. Finally, an important open issue remains the detection of malicious PDF documents whose exploitation techniques do not rely on JavaScript.

Acknowledgements

The authors wish to acknowledge finantial support by the Heiseberg Fellowship of the Deutsche Forschungsgemeinschaft (DFG) and by the German Federal Office for Information Security.

9. REFERENCES

[1] Internet security threat report. Symantec, 2010.

[2] APSB11-03. http://www.adobe.com/support/security/-bulletins/apsb11-03.html.

[3] APSB11-08. http://www.adobe.com/support/security/-bulletins/apsb11-08.html.

[4] M. Barreno, B. Nelson, A. Joseph, and J. Tygar. The security of machine learning. *Machine Learning*, 81(2):121–148, 2010.

[5] D. Canali, M. Cova, G. Vigna, and C. Kruegel. Prophiler: a fast filter for the large-scale detection of malicious web pages. In *International Conference on World Wide Web (WWW)*, pages 197–206, 2011.

[6] C.-C. Chang and C.-J. Lin. *LIBSVM: a library for support vector machines*, 2001. Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

[7] M. Cova, C. Kruegel, and G. Vigna. Detection and analysis of drive-by-download attacks and malicious JavaScript code. In *International Conference on World Wide Web (WWW)*, pages 281–290, 2010.

[8] C. Curtsinger, B. Livshits, B. Zorn, and C. Seifert. ZOZZLE: Fast and precise in-browser javascript malware detection. In *USENIX Security Symposium*, 2011. to appear.

[9] A. Dewald, T. Holz, and F. Freiling. ADSandbox: sandboxing JavaScript to fight malicious websites. In *Symposium on Applied Computing (SAC)*, pages 1859–1864, 2010.

[10] M. Engelberth, C. Willems, and H. T. MalOffice – analysis of various application data files. In *Virus Bulletin International Conference*, 2009.

[11] B. Feinstein and D. Peck. Caffeine Monkey: Automated collection, detection and analysis of malicious JavaScript. In *Black Hat USA*, 2007.

[12] K. Itabashi. Portable document format malware. Symantec white paper, 2011.

[13] W.-J. Li, S. Stolfo, A. Stavrou, E. Androulaki, and A. Keromytis. A study of malcode-bearing documents. In *Detection of Intrusions and Malware & Vulnerability Assessment (DIMVA)*, pages 231–250, 2007.

[14] J. Nazario. PhoneyC: a virtual client honeypot. In *USENIX Workshop on Large-scale Exploits and Emergent Threats (LEET)*, 2009.

[15] PDF Reference. http://www.adobe.com/devnet/pdf/pdf_reference.html, 2008.

[16] M. Polychronakis, K. Anagnostakis, and E. Markatos. Comprehensive shellcode detection using runtime heuristics. In *Annual Computer Security Applications Conference (ACSAC)*, pages 287–296, 2010.

[17] N. Provos, P. Mavrommatis, M. Abu Rajab, and F. Monrose. All your iFRAMEs point to us. In *USENIX Security Symposium*, pages 1–16, 2008.

[18] P. Ratanaworabhan, B. Livshits, and B. Zorn. NOZZLE: A defense against heap-spraying code injection attacks. In *USENIX Security Symposium*, pages 169–186, 2009.

[19] K. Rieck, T. Krüger, and A. Dewald. Cujo: Efficient detection and prevention of drive-by-download attacks. In *Annual Computer Security Applications Conference (ACSAC)*, pages 31–39, 2010.

[20] K. Rieck and P. Laskov. Linear-time computation of similarity measures for sequential data. *Journal of Machine Learning Research*, 9:23–48, 2008.

[21] Z. Shafiq, S. Khayam, and M. Farooq. Embedded malware detection using markov n-grams. In *Detection of Intrusions and Malware & Vulnerability Assessment (DIMVA)*, pages 88–107, 2008.

[22] R. Sommer and V. Paxson. Outside the closed world: On using machine learning for network intrusion detection. In *IEEE Symposium on Security and Privacy*, pages 305–316, 2010.

[23] D. Tax and R. Duin. Support vector data description. *Machine Learning*, 54:45–66, 2004.

[24] Z. Tzermias, G. Sykiotakis, M. Polychronakis, and E. Markatos. Combining static and dynamic analysis for the detection of malicious documents. In *European Workshop on System Security (EuroSec)*, 2011.

[25] Y.-M. Wang, D. Beck, X. Jiang, R. Roussev, C. Verbowski, S. Chen, and S. King. Automated web patrol with strider honeymonkeys: Finding web sites that exploit browser vulnerabilities. In *Network and Distributed System Security Symposium (NDSS)*, 2006.

[26] C. Willems, T. Holz, and F. Freiling. CWSandbox: Towards automated dynamic binary analysis. *IEEE Security and Privacy*, 5(2):32–39, 2007.

[27] J. Wolf. OMG WTF PDF. Chaos Communication Congress (CCC), Dec. 2010.

Nexat: A History-Based Approach to Predict Attacker Actions

Casey Cipriano
UCSB
CS Department
Santa Barbara, CA, USA
ccipriano@gmail.com

Ali Zand
UCSB
CS Department
Santa Barbara, CA, USA
zand@cs.ucsb.edu

Amir Houmansadr
UIUC
ECE Department
Urbana, IL, USA
ahouman2@illinois.edu

Christopher Kruegel
UCSB
CS Department
Santa Barbara, CA, USA
chris@cs.ucsb.edu

Giovanni Vigna
UCSB
CS Department
Santa Barbara, CA, USA
vigna@cs.ucsb.edu

ABSTRACT

Computer networks are constantly being targeted by different attacks. Since not all attacks are created equal, it is of paramount importance for network administrators to be aware of the status of the network infrastructure, the relevance of each attack with respect to the goals of the organization under attack, and also the most likely next steps of the attackers. In particular, the last capability, attack prediction, is of the most importance and value to the network administrators, as it enables them to provision the required actions to stop the attack and/or minimize its damage to the network's assets. Unfortunately, the existing approaches to attack prediction either provide limited useful information or are too complex to scale to the real-world scenarios.

In this paper, we present a novel approach to the prediction of the actions of the attackers. Our approach uses machine learning techniques to learn the historical behavior of attackers and then, at the run time, leverages this knowledge in order to produce an estimate of the likely future actions of the attackers. We implemented our approach in a prototype tool, called *Nexat*, and validated its accuracy leveraging a dataset from a hacking competition. The evaluations show that *Nexat* is able to predict the next steps of attackers with very high accuracy. In particular, *Nexat* achieves a 94% accuracy in predicting the next actions of the attackers in our prototype implementation. In addition, *Nexat* requires little computational resources and can be run in real-time for instant prediction of the attacks.

Categories and Subject Descriptors

C.2.0 [**Computer-Communication Networks**]: General—*Security and protection*; C.2.3 [**Computer-Communication Networks**]: Network Operations—*Network monitoring*

General Terms

Algorithms, Design, Security

Keywords

Attack prediction, situation awareness, machine learning

1. INTRODUCTION

Computer networks are constantly under various attacks. Network administrators deploy intrusion detection systems (IDS) in order to detect the occurrence of the *events* that may be part of an attack [1, 19, 24]. Unfortunately, while intrusion detection systems provide useful information about the attack-related events, they provide little information about the network's *situation* before and after any of the attacks. In order to better provision and respond to the attacks a network administrator needs the answers to the following questions: how are different events related to an attack? what is the impact of an attack on the network? and, the most important, what are the most likely next actions of the attackers? The answers to these questions and other similar questions provide a high-level understanding of the security situation of the computer networks, also referred to as *situation awareness*. Situation awareness has long been referred to as an important and critical aspect for cyber-defense [7, 6, 10, 23, 3, 26, 20, 14]. Endsley [6] proposes a general definition for situation awareness as "the perception of the elements of the environment within a volume of time and space, the comprehension of their meaning, and the projection of their status in the near future." With respect to the cyber-defense scenario, situation awareness tries to contextualize cyber-attacks with respect to the network infrastructure and the mission flows in order to enable a network administrator to prioritize certain preventative measures and also determine the severity of different attacks [25]. Furthermore, situation awareness can provide estimates of the likely next steps of an attacker, assisting a network administrator in using appropriate preventative approaches instead of just reactive ones [21, 22].

Predicting the next actions of the attackers is an important, yet difficult, aspect of situation awareness. Existing approaches for attack prediction either provide limited predictive information [15, 21, 13] or are complex and can not be utilized for large-scale scenarios [18, 8, 22, 28]. As an example, Qin et al. uses Granger causality analysis in order to compute precursor alerts [21]. This approach provides limited accuracy in attack prediction as its performance is very sensitive to the presence of unrelated background alerts. As another example, Geib et al. provide a plan prediction scheme that uses pre-determined attacker plans in order to predict future attacks [8]. This approach is not scalable, as it does not perform in a fully automated manner and requires continuous inputs from the network administrators in order to perform the predictions. Therefore, there is the need for effective situation awareness tools that provide attack prediction capabilities that are not only accurate, but also scalable to large-scale networks.

In this paper, we leverage *machine learning* to predict the next steps of attacks based on histories of previous attacks. We implemented our approach in a tool called *Nexat*, which is the focus of this paper. *Nexat* uses machine learning techniques to *learn* the past behavior of the attackers, and then uses this knowledge to predict the next steps of future attacks in real-time. More specifically, *Nexat* consists of three operational phases: the data extraction phase, the training phase, and the prediction phase. During the data extraction phase, *Nexat* extracts some information from the alerts previously generated by intrusion detection systems. *Nexat* uses this extracted information during the training phase in order to generate a knowledge base about the attackers' behavior. Finally, *Nexat* uses the generated models of the network and the attackers to make predictions on the next steps of a live stream of attack alerts. The attack predictions provided by *Nexat* can be used by decision makers, e.g., network administrators, to devise efficient defensive actions and to estimate the impact of a future attack on the critical assets of the organization.

We evaluated the performance of *Nexat* using a prototype implementation on a unique database of attack alerts. The alert database is collected by an intrusion detection system during a large-scale hacking competition, where a number of hacker teams tried to compromise the same target network by taking multiple steps. Our evaluation shows that *Nexat* is able to learn the attacking behavior of the attackers and use it to predict their future actions efficiently. In particular, by learning the attack histories of 32 teams *Nexat* was able to predict the actions of another hacker team with an average accuracy of 94%.

The rest of this paper is organized as follows: in Section 2, we give a detailed description of our attack prediction algorithm, *Nexat*. Section 3 evaluates a prototype implementation of *Nexat* by discussing its prediction efficiency. In Section 4 we discuss some of the related work. Finally, the paper is concluded in Section 5.

2. NEXAT: HISTORY-BASED ATTACK PREDICTION

In this section, we describe the design of *Nexat*, a tool that we develop for predicting attackers' likely next steps. *Nexat* uses a machine learning technique to predict the attacker actions based on the activity history of the attackers. More specifically, *Nexat*'s operation can be divided into three main phases: the data extraction phase, the training phase, and the prediction phase. Before describing these steps, we first provide a formal definition of some of the terms used later. Also, Table 1 summarizes all of the elements used in the design of *Nexat*, e.g., lists and hash tables, along with their symbols and definitions.

DEFINITION 1. **(Alert)** *An* alert *is a set of information about a suspicious network event being observed and reported by an IDS system.*

The set of information included in an alert depends on the type of the reporting IDS system; common examples of alert information are the "*source address*" and the "*destination address*" of the network flow associated with the event, and the time of its occurrence.

DEFINITION 2. *(***Attack session***) An* attack session *is a sequence of alerts whose properties satisfy certain conditions. We define an alert A to be part of an attack session S if at least one of these properties holds:*

- *the destination of alert A is the same as the destination of a previous alert in S,*
- *the source of alert A is the same as the source of a previous alert in S,*
- *the source of alert A is a destination of a previous alert in S.*

In addition, A should lie within a time window of w seconds from the last alert in S.

The properties used in specifying the attack sessions are meant to capture three common attack patterns, namely, *one-to-many*, *many-to-one*, and *island-hopping*. A single source attacking many unique targets is classified as a one-to-many pattern, whereas multiple sources attacking a single target is referred to as a many-to-one attack. Finally, island-hopping occurs when an attacker uses a previously-compromised target to attack another target. By looking for each of these attack patterns we establish a basis for finding all of the alerts that belong to a specific attack.

The *data extraction* phase takes as input a training set containing a history of previously-observed alerts. Using these alerts, *Nexat* extracts a list of all of the possible attack sessions by grouping together the alerts as will be described later. The rational behind this classification is that alerts pertaining to the same attack session are very likely to be part of the same attack, and, hence, they are very likely to happen simultaneously in the future. During the *training phase*, *Nexat* analyzes the extracted attack sessions to identify the co-occurrence relationship amongst different alerts being involved in these sessions. This results in a knowledge base for *Nexat* which is used during the *prediction phase* to probabilistically predict the next actions of an attacker, once a live stream of alerts are being observed by an intrusion detection system. In the following, we provide the detailed description of these three phases.

2.1 Data Extraction Phase

In this phase, *Nexat* extracts attack sessions out of a list of alerts that are reported by one or more IDSs. This is done in two steps: generating the list of alerts, and extracting attack sessions from the alerts list.

Generating the alerts list

The current implementation of *Nexat* takes Snort alerts[1] as input and extracts four of its attributes, i.e., the name of the attack, the source address, the destination address, and the time the alert was generated. Table 2 shows some sample alerts generated by a Snort IDSs. *Nexat* constructs and maintains an *alerts list* that keeps the alerts collected by the IDSs in an ascending order of time. We assume that multiple IDSs are time-synchronized. Table 3 shows a sample alerts list. For the sake of simplicity, we use pseudonyms for the name of the alerts and also for the source and destination IP addresses.

Extracting the attack sessions

For any alert A in an alerts list AL, *Nexat* generates an attack session S initiating from A as defined in Definition 2. More specifically, in order to add an alert A_1 to an existing attack session, one

[1]The code can easily be extended to take inputs from other IDSs as well.

Table 1: Different elements used in *Nexat*.

Name	Symbol	Description
Alert	A	An alert as defined in Definition 1.
Alerts list	AL	A list that contains the list of alerts collected from an IDS in ascending order of time.
Attack session	S	An attack session as defined in Definition 2.
List of attack sessions	H	A hash table keyed by alerts, containing the corresponding attack sessions.
Trained dataset	T	A hash table that contains the empirical probabilities trained during the training phase.
Power-set of an alert	$P(A)$	The power-set corresponding to an alert as defined in the text.
Results list	$R(T, I)$	A hash table that provides predictions on the next alert of an attack stream of I based on the trained dataset T.

Table 2: Example alerts generated by an Snort IDS.

Pseudonym	Description
A	DNS named authors attempt
B	(http_inspect) OVERSIZE REQUEST-URI DIRECTORY
C	SNMP trap TCP
D	WEB-CGI bb-hist.sh access
E	MISC source port 53 to <1024
F	BACKDOOR Wordpress backdoor feed.php code execution attempt

Table 3: An example list of alerts. The source and destination addresses are replaced by pseudonyms of the hosts.

Alert name	Attribute		
	Source address	Destination address	Time
A	9	6	1
G	2	4	2
B	8	7	4
D	7	1	7
K	3	4	8
J	2	5	9
E	1	6	10

of these three requirements should be met: A_1's destination address is the same as the destination of an alert existing in S, A_1's source is the same as the source address of an alert in S, or A_1's source is the same as the destination address of an alert in S. *Nexat* uses a sliding time window, w, for performing such correlation. This algorithm, `GetSession(A∈ AL,AL)`, is depicted in Algorithm 2.1. `GetSession(A∈ AL,AL)` goes over an alerts list AL, being sorted in ascending order of time, to find all of the existing attack sessions corresponding to an alert $A \in AL$. For any instance of such an attack session, S, `GetSession` performs the following steps:

- For the attack session S, `GetSession` creates and maintains two lists $SourceList(S)$ and $TargetList(S)$, which contain the possible source address and destination addresses of the future alerts of S, respectively.
- `GetSession` traverses the alerts list AL in descending order of time starting from A in order to find and append related alerts to S. For any alert A_1 which lies in a window w from the last alert in S, `GetSession` adds it to S if either A_1's source is in $SourceList(S)$ or A_1's destination is in $TargetList(S)$.
- For any alert A_1 that is added to an attack session S, the destination of A_1 is added to $TargetList(S)$ and the source of A_1 is added to both $SourceList(S)$ and $TargetList(S)$ lists (this enables *Nexat* to find the three possible types of attack, namely, one-to-many, many-to-one, and island-hopping, as mentioned before).
- `GetSession` closes S if it does not find another alert from AL that satisfies the mentioned properties.

Nexat maintains a list of attack sessions, H, which is a hash table that is indexed by alert names. H stores lists of attack session sets corresponding to each alert. Upon finding an attack session S for an alert $A \in AL$, as described above, *Nexat* appends S to the line of H corresponding to A, i.e., $H[A]$. Once this is done for all of the alerts in AL the hash table H will hold a set of attack sessions for each alert in the AL. This algorithm is sketched in Algorithm 2.2, where the $GetSession(A, AL)$ algorithm evaluates the attack sessions for an alert A, as detailed before. The hash table H will serve as the input to the training phase, as described later. Table 4 shows a sample hash table H.

Algorithm 2.2: DATAEXTRACTIONPHASE(AL)

```
$H \leftarrow hashtable()$
**for each** $A \in AL$
  **do** $H[A].append($GETSESSION$(A, AL))$
```

Algorithm 2.1: GETSESSION($A \in AL, AL$)

```
$reversedList \leftarrow AL.reverse()$      % $X \leftarrow Y$ assigns the value of $Y$ to $X$
$attackSessionSet \leftarrow set()$
$TargetList \leftarrow set()$
$SourceList \leftarrow set()$
$lastAlert \leftarrow A$
$TargetList.append([lastAlert.target, lastAlert.source])$
$SourceList.append(lastAlert.source)$
for each $previousAlert \in reversedList$
  do
    if $previousAlert.time > lastAlert.time - w$
      then
        if ($previousAlert.target \in TargetList$
            or $previousAlert.source \in SourceList$)
          then
            $S.append(previousAlert.name)$
            $lastAlert \leftarrow previousAlert$
            $TargetList.append([previousAlert.source, previousAlert.target])$
            $SourceList.append(previousAlert.source)$
    else
      break
return ($S$)
```

Table 4: A sample list of attack sessions, H.

Alert Name	Attack sessions
E	{A,B,D}, {A,B}, {A,C}, {B,D}, {B}
F	{B,C,D}, {B,C}, {C}
...	...

An example run of the data extraction phase

For the sake of clarity, here we mention a sample run of the data extraction phase. Let us suppose that *Nexat* received as input an alerts list AL as shown in Table 3. As can be seen, AL contains the name of the alerts along with their corresponding sources and destination addresses, and the time when they occurred. We show how *Nexat* proceeds to find the attack sessions for the last alert of AL, i.e., E, for a time window size of $w = 5sec$.

Nexat traverses the AL list backwards, starting with alert J. Since J does not share any source or destination address similarities with E, it is not added it to the attack session S. The situation is the same for alert K, since there is no shared destination or source information with E. However, as can be seen, the destination address of the alert D is the same as the source address of the last alert in S, i.e., E. In fact, this can be an indication of an island-hopping attack. Since the time difference between the alerts D and E (the last alert in S) is less than the time window $w = 5sec$ the alert D is appended to the attack session S. In addition, the source address of D is added to $SourceList(S)$ and $TargetList(S)$, and its destination address is added to $TargetList(S)$. Similarly, the alerts B and A are appended to S and their source and destination addresses are added to the corresponding lists. When *Nexat* reaches the end of the AL list (or the time window w from the last alert has elapsed), `GetSession(`E`,AL)` closes the attack session S and appends it to the line indexed E of the list of attack sessions, e.g., $H[E]$. In this example, the attack session $S = \{A, B, D\}$ is appended to the list of attack sessions shown in Table 4. The following is a real-world sample of an attack session that we observed in our evaluations:

{"ICMP PING NMAP" , "(http_inspect) BARE BYTE UNICODE ENCODING" , "WEB-PHP PHP function CRLF injection attempt" , "WEB-MISC .htpasswd access" , "(http_inspect) DOUBLE ENCODING ATTACK" , "WEB-MISC encoded cross site scripting attempt" , "SHELLCODE x86 NOOP"}.

2.2 Training Phase

Nexat uses the list of attack sessions, i.e., the hash table H, generated during the data extraction phase to train about the co-occurrence of the alerts, i.e., to determine which alerts are more likely to occur in the same attack session. The training algorithm is depicted in Algorithm 2.3.

Algorithm 2.3: TRAININGPHASE(H)

```
$T \leftarrow hashtable()$
for each $A \in H$
  do
    $U(A) \leftarrow list()$
    for all $S \in H[A]$
      do $U = U \cup S$
    for each $x \in$ POWERSET$(U) \backslash \{\emptyset\}$
      do
        $o \leftarrow 0$
        for each $S \in H[A]$
          do if $x \subseteq S$
            then $o \leftarrow o + 1$
    $T[x].append(\{o, A\})$
$total \leftarrow hashtable()$
for each $y \in T$
  do
    $total[y] \leftarrow 0$
    $total \leftarrow total + T[y].o$
for each $y \in T$
  do $T[y].o \leftarrow T[y].o/total[y]$
```

During the training phase, the `TrainingPhase` algorithm uses the list of attack sessions H, created in the data extraction phase, to generate a *trained dataset* T. The trained dataset T is a hash table indexed by the set of alerts. For any set of alerts $x \in T$, T keeps a list of pairs (p, A) such that p is the empirical probability that x is followed by the alert A. The generation of T is further described later. The training algorithm of *Nexat* is performed as follows:

Table 5: A sample trained dataset T.

(a) Before normalization

{A}	{3, E}
{B}	{4, E}, {2, F}
{C}	{1, E}, {3, F}
{D}	{2, E}, {1, F}
{AB}	{2, E}
{AC}	{1, E}
{AD}	{1, E}
{BC}	{2, F}
{BD}	{1, E}, {1, F}
{CD}	{1, F}
{ABC}	{Ø}
{ABD}	{1, E}
{ACD}	{Ø}
{BCD}	{1, F}

(b) After normalization

{A}	{1, E}
{B}	{.66, E}, {.33, F}
{C}	{.25, E}, {.75, F}
{D}	{.66, E}, {.33, F}
{AB}	{1, E}
{AC}	{1, E}
{AD}	{1, E}
{BC}	{1, F}
{BD}	{.5, E}, {.5, F}
{CD}	{1, F}
{ABC}	{Ø}
{ABD}	{1, E}
{ACD}	{Ø}
{BCD}	{1, F}

- For each alert $A \in H$, `TrainingPhase` evaluates the union set of all of the attack sessions that H keeps for A. As an example, if for an alert A the table H returns two attack sessions of $\{B, C\}$ and $\{B, F\}$ the union set for A would be $U(A) = \{B, C, F\}$. In fact, the union set $U(A)$ returns a set of possible precursor alerts for the alert A.

- `TrainingPhase` evaluates $P(A)$, which is the power-set of $U(A)$ excluding the empty set $\{\emptyset\}$ and limited to a maximum size of k. As an example, for the $U(A) = \{B, C, F\}$ mentioned the corresponding power-set is $P(A) = \{\{B\}, \{C\}, \{F\}, \{B, C\}, \{B, F\}, \{C, F\}, \{B, C, F\}\}$. The limit k is used to bound the computational complexity of *Nexat* and is set to 3 in our implementation.

- For any set $x \in P(A)$, `TrainingPhase` evaluates $N(x, A)$ to be the number of attack sessions, $S \in H[A]$ that are a superset of x, i.e., $N(x, A) = ||\{S|S \in H(A) \& x \subseteq S\}||$. If $N(x, A) > 0$ the pair $(N(x, A), A)$ is appended to the line x of the hash table T, i.e., $T[x]$.

- Finally, `TrainingPhase` normalizes the entries of the T table by dividing each $N(\cdot, \cdot)$ by the sum of all of the $N(\cdot, \cdot)$ entries in the same row of table T.

Table 5 shows a sample T hash table before and after normalization. For any set $x \in T$, a normalized pair of (p, A) indicates that with probability p the alert A proceeds the set of alerts in x. This is used in the prediction phase to predict the next steps of an attack, as described later.

It should be mentioned that limiting the size of the power-sets to $k = 3$ puts a constraint on the storage and the computational resources required by *Nexat*. Increasing k improves the training process of *Nexat* at the cost of exponentially increasing the required resources.

An example run of the training phase

We continue the example mentioned in Section 2.1 for the training phase of *Nexat*. The data extraction phase of *Nexat* generates the list of attack sessions H that is shown in Table 4. As can be seen, H keeps the attack session sets of $\{A, B, D\}, \{A, B\}, \{A, C\}, \{B, D\}, \{B\}$ for the alert E, and the attack session sets of $\{B, C, D\}, \{B, C\}, \{C\}$ for the alert F. So, the union set corresponding to E can be evaluated as $U(A) = \{A, B, C, D\}$, resulting in the corresponding powerset to be

$$P(E) = \{\{A\}, \{B\}, \{C\}, \{D\}, \{AB\}, \{AC\}, \{AD\}, \{BC\}, \{BD\}, \{CD\}, \{ABC\}, \{ABD\}, \{ACD\}, \{BCD\}\}$$

As mentioned before, the powerset $P(\cdot)$ excludes the empty set and is limited to subsets with maximum size of $k = 3$. One can similarly find the powerset corresponding to the alert F as

$$P(F) = \{\{B\}, \{C\}, \{D\}, \{BC\}, \{BD\}, \{CD\}, \{BCD\}\}$$

`TrainingPhase` uses these powersets to generate the trained dataset T as described before. Table 5 shows the generated T before and after normalization.

2.3 Prediction phase

In the prediction phase, *Nexat* uses the trained dataset T generated during the training phase to make predictions on the live streams of attacking events. More specifically, for each alert in a live stream of alerts I returned by one or more IDSs, *Nexat* predicts the most likely forthcoming alerts.

The prediction algorithm of *Nexat* is illustrated in Algorithm 2.4. `PredictionPhase` uses a *results list*, $R(T, I)$, to store the predictions performed on I using the trained dataset of T. $R(T, I)$ is a hash table indexed by the name of the alerts, where for each alert it stores a list of *result pair*s in the form of (p, ℓ) where p is the probability and ℓ is the length of a set of alerts. The `PredictionPhase` algorithm works as follows:

- Suppose that A_r is the last received alert in the live stream of alerts I. `PredictionPhase` evaluates the attack session S_r corresponding to A_r using the `GetSession(`A_r`,`I`)` algorithm mentioned in Algorithm 2.1.

- `PredictionPhase` also evaluates P_r, the limited powerset of S_r as defined before (excluding the empty set and limited to size $k = 3$).

- For any $x \in P_r$ and for any $r \in T[x]$, `PredictionPhase` appends the result pair of $(r.p, length(x))$ to the line x of hash table R, i.e., $R[x]$. $r.p$ is the probability attribute of r, as defined in Section 2.2, and $length(x)$ returns the length of the set x. This identifies all of the records in the trained dataset that are related to the current alert.

- Finally, `PredictionPhase` normalizes the probability attribute in the result pairs of the R table. In order to do this, *Nexat* evaluates the weighted sum of the probabilities for each row (corresponding to alert A) of the table as $Sum_A = \sum_{i=1}^{Num_A} p_i \times \ell_i$, where Num_A is the number of result pairs corresponding to alert A in R. `PredictionPhase`, then, divides all of the probability attributes of the result pairs by Sum_A to produce the normalized weighted probabilities.

Algorithm 2.4: PREDICTIONPHASE(T, I)

```
$R \leftarrow hash()$
for each $x \in$ POWERSET(GETSESSION($A_r, I$))$\backslash\emptyset$
  do { for each $r \in T[x]$
         do $R[r].append(\{r.p, length(x)\})$
for each $A \in R$
  do { $sum \leftarrow 0$
       for each $record \in R[A]$
         do $sum \leftarrow sum + record.p * record.\ell$
       $R[A] \leftarrow sum$
$sum \leftarrow 0$
for each $A \in R$
  do $sum \leftarrow sum + R[A]$
for each $A \in R$
  do $R[A] \leftarrow R[A]/sum$
```

Nexat uses the weighted probabilities calculated above to predict the future alerts. In fact, by weighting the probabilities we give more significance to the longer attack sessions from the trained dataset. Table 6 shows a sample results list R.

An example run of the prediction phase

We will continue with the data provided from the training phase example. `PredictionPhase` uses the trained dataset T from the training phase to make predictions on a live stream of alerts I. Suppose that running the `GetSession` algorithm on the most recent alert of the stream I returns the set $S_I = \{B, C, D\}$. The power-set of S_I, as defined before, results in the set

$$\{\{B\}, \{C\}, \{D\}, \{BC\}, \{BD\}, \{CD\}, \{BCD\}\}$$

. `PredictionPhase` extracts all of the corresponding records from T and imports them into the results list R, as shown in the Table 6. As can be seen, the entries of this hash table are the result pairs in the form of (p, ℓ), where p is the probability of the subset and ℓ is the length of the subset. The columns of the table show the corresponding subsets for the sake of clarity. The table also shows the weighted sum for the alerts E and F. Using the wighted sums *Nexat* provides a predicted probability for any of the alerts to be the next alert of I. As the table shows, for the mentioned example *Nexat* provides a 78.5% prediction of F being the next alert, and a 21.5% predication of E being the next alert.

3. EVALUATION

In this section, we evaluate the performance of the *Nexat* attack prediction tool described in this paper. We run a prototype implementation of *Nexat* over a large database of attack alerts gathered during the *2008 UCSB International Capture The Flag* (iCTF) hacking competition [2], held on December 2008. The alert database contains 248,783 alerts that are generated by a snort intrusion detection system during the competition. A total number of 800 attackers from different educational institutions participated in the iCTF competition in the form of 40 hacking teams.

An identical copy of the same multi-host network was dedicated to each team, and the goal of the competition for the teams was to compromise their dedicated networks through an arbitrary number of attacking steps. The network was monitored by both signature-based and anomaly-based intrusion detection systems. The competing teams were allowed to use any hacking technique, however, they would loose points whenever their activities we detected by the intrusion detection systems. More information about the iCTF 2008 competition including the competition rules, the architecture of the competition networks, and the list of the competitors is available online [2].

We use the database of alerts described above to evaluate the prediction performance of *Nexat*: we use the alerts generated as a result of attacking actions of one or more teams to train *Nexat*, and then use the trained *Nexat* to predict the next actions of other teams. In fact, this resembles the real-world scenario for attack prediction: a network administrator aims at predicting future attacks by having access to a history of attacks performed against the *same* network infrastructure. The conducted evaluations are described in the following.

Before explaining the experiments we define the accuracy metric that is used for the evaluations.

DEFINITION 3. **Accuracy:** *We define the accuracy of* Nexat *in predicting an stream of alerts, I, to be the mean of the probabilities that* Nexat *predicts for all of the observed alerts. More specifically, if* Nexat *predicts the next alerts of I with probabilities p_i ($i = 1, .., n$), the accuracy of* Nexat *is $\sum_{i=1}^{n} p_i/n$.*

3.1 Experiment 1: trained by one team

In this experiment, *Nexat* is trained using the Snort alerts corresponding to one of the hacking teams, and then the trained models are used to predict the actions of the other teams. To do this, we extracted the alerts corresponding to any of the teams from the mentioned alerts database. Teams who produced less than 100 alerts were ignored as this would not be sufficient data to provide for any meaningful training. This filtering process left us with 33 unique teams, each generating between 100 and 20000 alerts. This resulted into 32^2 series of predictions, i.e., we choose one team for training and try to predict actions of the other 32 teams.

Figure 1 depicts the results of the experiments for a time window of $w = 5sec$. Each point on the figure shows the accuracy of *Nexat*, as defined in Definition 3, in predicting the actions of one hacker team, being trained by the attack history of another team. The horizontal axis of the figure shows the name of the team that *Nexat* uses for training. The names assigned to the teams are not their actual names, but their ranking at the end of the competition, e.g., team 3 gained the third place in the competition. The empty columns of the figure correspond to the teams that produced less than 100 alerts, hence were not used to train *Nexat*. For any of the hacker teams used for training *Nexat*, the figure also shows the average accuracy of *Nexat* in predicting the actions of all the other teams. As can be seen, in 22 cases (out of a total of 33 cases) the mean accuracy of *Nexat* in predicting the actions of the other teams is more than 60%. This is significantly promising, because *Nexat* uses the history of *only* one hacker team during the training phase. We observe that in a few cases, using a single team for training *Nexat* results in very poor prediction performance, e.g., in the case of teams ranked 13 and 24. We call such teams *bad predictor* teams. In fact, the bad predictor teams take hacking actions that are different from the rest of the teams. We can see that most of the bad predictor teams are placed in the lower two-third of the competition's ranking table, i.e., only one of the first 12 teams is a bad predictor. We conclude that the more skilled a hacker team is the better prediction performance is achieved by using that team for training *Nexat*.

We also use a force-based algorithm to show the clusters of the hacker teams based on the results of the predictor, as shown in Figure 2. Each circle in the figure represents one of the hacker teams, where the number inside the circle is the rank of the team in the competition. Two teams are connected in the graph if the accuracy of *Nexat* in predicting each of them is more than 0.5, by using the

Table 6: A sample result list R.

Alerts	{B}	{C}	{D}	{BC}	{BD}	{CD}	{BCD}	Weighted sum	Predicted probability
E	(.66, 1)	(.25, 1)	(.66, 1)	(0,2)	(.5, 2)	(0,2)	(0,3)	2.583	0.215
F	(.33, 1)	(.75, 1)	(.33, 1)	(1, 2)	(.5, 2)	(1, 2)	(1, 3)	9.416	0.785

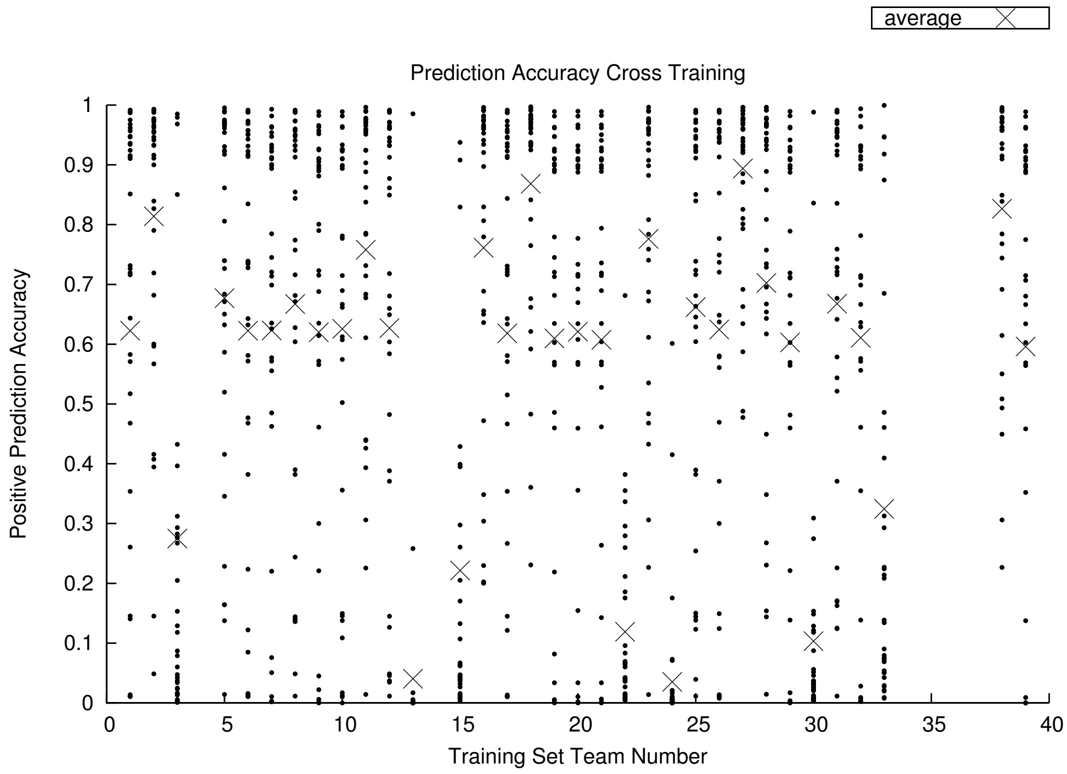


Figure 1: The prediction accuracy of *Nexat* using only one team for training.

other one for training. As can be seen, most of the teams are well connected in the graph, showing a high correlation between their actions. The teams which are less connected to the graph represent the bad predictor teams, as discussed before.

3.2 Experiment 2: trained by all the other teams

The results from the first experiment show how important a quality training set is to making accurate predictions. The teams whose training sets proved to be poor, likely had little to no overlap with the vast majority of teams in the competition, with respect to their attack behavior. In this experiment we try to improve the prediction accuracy of *Nexat* by using the histories of multiple hacker teams to train *Nexat*. This is unlike the first experiment where only one team's history was used for training.

Figure 3 shows the prediction accuracy of *Nexat* for each team, when the histories of all the other teams are used to train *Nexat*. In other words, each point on the graph shows the accuracy of *Nexat* in predicting the team shown in the horizontal axis, being trained by the other 32 teams. As can be seen, using the actions of multiple teams to train *Nexat* drastically improves its prediction performance. In particular, all the prediction accuracies are more than 80%, and the average prediction accuracy is 94%. This is due to the fact that the training set has a much larger coverage of unique alerts, since it could combine the strategies of several different teams.

3.3 Effect of the time window on the results

As mentioned before, in the training phase of *Nexat* a time window of w is used in order to divide each complex compromise activity into distinct sessions. The value of w impacts the prediction accuracy of *Nexat* in different ways. By using a large value for w, *Nexat* can collect more of the alerts that correspond to the same complete attack, hence improving its training phase. This, however, increases the computational complexity of *Nexat* by generating more sets of alerts. Using a properly sized time window gives us a balance of prediction accuracy and computational complexity.

Figure 4 shows the average prediction accuracy of *Nexat* being trained on the history of team 25, for different values of the w parameter in the logarithmic scale. As can be seen, increasing w drastically improves the prediction accuracy for $w \leq 10sec$, however, it does not change significantly for the larger values of w. Considering the impact of w on the computational complexity, we choose $w = 5$ in our implementation, as it makes a reasonable tradeoff between the complexity and the accuracy of *Nexat*.

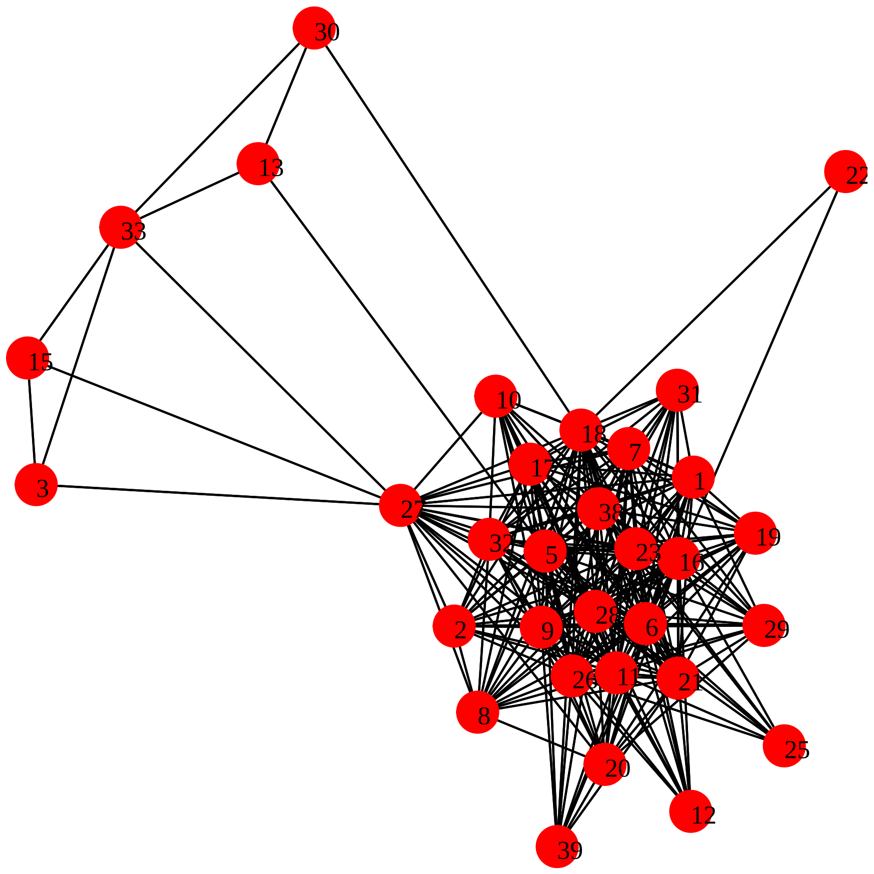


Figure 2: Clustering the hacker teams based on the prediction accuracy of *Nexat* (only one team is used for training).

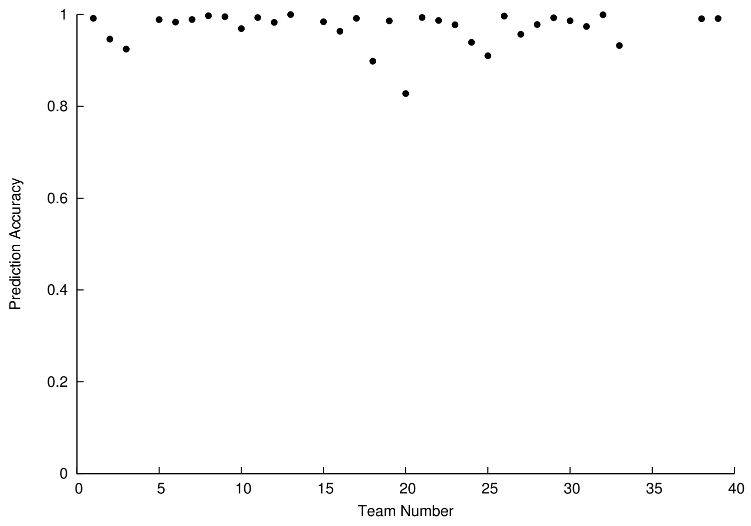


Figure 3: The prediction accuracy of *Nexat* using 32 teams for training.

3.4 Nexat's resources

We ran the prototype implementation of *Nexat* on a laptop with a 2.53GHz processor, 4GB memory, and 100GB disk space. Our measurements show that *Nexat* is able to predict the attacks with very reasonable resourcces, making it suitable to be deployed in real-time scenarios. *Nexat* performed the training phase for each team in about 1.7 seconds, using around 30 MB of RAM for each team. This resulted in less than one minute to train all of the com-

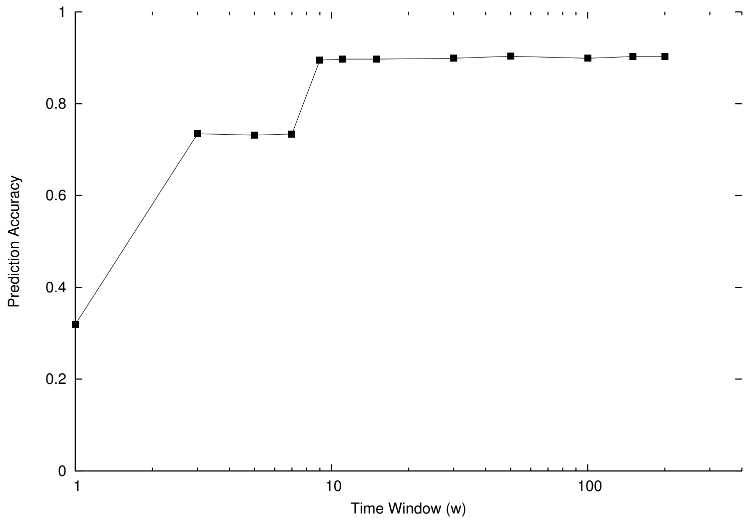


Figure 4: The effect of the time window w on the prediction accuracy.

peting teams. During the prediction phase, *Nexat* ran the data from each team against any of the profiles in about 2.4 seconds, using an average memory of 32MB. This resulted in a total prediction time of about one minute and a half for each of the teams. We expect that a customized implementation of *Nexat* on a powerful network monitoring device will result in much faster operation of *Nexat*.

4. RELATED WORK

Several different approaches have been taken by researchers in order to analyze security alerts. Several researchers have considered alert correlation by grouping low-level alerts together in order to find cohesive groups or precursor of alerts [12, 11, 4, 5, 27, 16, 17]. As they rely on different levels of predefined knowledge of attack conditions and consequences, most of such approaches have limited effectiveness. In other words, they are not able to provide valid alert correlations for unknown attacks or unknown attack relations. Plan recognition has been a common approach to predicting attacker behavior [8]. Plan recognition approaches use predetermined attacker plans in order to predict future attacks. Plan recognition approaches, however, are not scalable as they require inputs from a security professional in order to perform their predictions. This is in contrast to *Nexat*, which performs the prediction in a fully automated manner using machine learning techniques.

Alternatively, some research use statistical tools in order to analyze the security alerts. Qin et al. look for attack scenarios by correlating the alerts and finding their causality relations [21]. More specifically, they classify the alerts into a number of clusters and present each cluster as a time series of alerts. Also, the resulting alerts are ranked based on the expert knowledge. Finally, Granger causality analysis is used on the time series of the top-ranked alerts in order to compute precursor alerts. The Granger causality test gives statistical predictions on whether a time series of alerts is an adequate predictor of another time series of alerts. Unfortunately, the performance of this approach is very sensitive to the presence of unrelated background alerts. For instance, running their tool on a 5-day sample of the 2000 DARPA Intrusion Detection Scenarios dataset results in about 94% true causality detection and 14% false causality detection. The false positives are even higher for DEFCON 9 traces due to the presence of more background alerts.

Qin and Lee conduct probabilistic inference using casual networks to recognize the attack plans and make some predictions on the attacks [22]. More specifically, the authors develop a graph-based technique in order to correlate isolated attack scenarios, extracted from low-level alert correlation, based on their relations in attack plans. This is followed by probabilistic inference to evaluate the likelihood of the attack goals, hence predict potential future attacks using the generated causal networks. The authors evaluate their algorithms on the dataset of attack alerts of the DARPA's Grand Challenge Program of 2003. The paper only provides sample attack predictions with average 0.7 chance of prediction. This is in contrast to the higher accuracies of *Nexat*, as described in this section. Another main issue with this approach is that it is not scalable as the size of the attack trees and the causal networks grows very rapidly with the size of the dataset. Wang et al. also use attack graphs for analyzing the alerts and making predictions [28]. The paper differs from previous work by using a queue graph instead of the timing windows to analyze the recent alerts, requiring less memory and speeding up the attack correlation tasks. The basic queue graph approach is also extended to a unified method to identify the missing alerts and to predict future alerts. The authors provide sample successful alert prediction but point out the possibilities of false positives and false negatives in practice due to incomplete or inaccurate domain knowledge. Moreover, this approach is sensitive to the timing inaccuracies of the reporting IDS systems.

Some research use game theory in order to predict the next actions of cyber attacks [13, 9]. In order to make the prediction, the game theoretic approach requires knowledge about all of the possible actions of the attackers against the defending system in order to predict the best actions of the attackers in different scenarios. This information is not usually available to the defending systems which degrades the applicability of this approach in many scenarios.

5. CONCLUSIONS

In this paper, we presented the design and implementation of *Nexat*, a tool for the prediction of attacker behavior. *Nexat* relies on machine-learning techniques to learn the attacker's behavior from previous alert histories and does not require any input from security professionals to make predictions. Also, since *Nexat*'s prediction is not order-dependent, it can handle minor timing errors associated with the reporting from intrusion detection systems. We validate the prediction accuracy of *Nexat* through a prototype implementation. The results show that by training *Nexat* on a large database of attacker behavior it is possible to perform attack prediction with an average accuracy of 94%. Future work will include researching alternative machine learning algorithms in order to further improve the accuracy of the predictions as well as lowering the computational complexity.

6. ACKNOWLEDGEMENTS

This work was supported by the ARO under grant W911NF-09-1-0553, the National Science Foundation (NSF) under grants CNS-0845559 and CNS-0905537.

7. REFERENCES

[1] IBM Internet Security Systems. `http://www.iss.net/`.

[2] The 2008 UCSB International Capture The Flag (iCTF). `http://ictf.cs.ucsb.edu/archive/iCTF_2008/index.html`, December 5th 2008.

[3] Paul Barford, Marc Dacier, Thomas G. Dietterich, Matt Fredrikson, Jon Giffin, Sushil Jajodia, Somesh Jha, Jason Li, Peng Liu, Peng Ning, Xinming Ou, Dawn Song, Laura Strater, Vipin Swarup, George Tadda, Cliff Wang, and John Yen. Cyber SA: Situational Awareness for Cyber Defense. In Sushil Jajodia, Peng Liu, Vipin Swarup, and Cliff Wang, editors, *Cyber Situational Awareness*, volume 46 of *Advances in Information Security*, pages 3–13. Springer US, 2010.

[4] F. Cuppens and A. Miege. Alert Correlation in a Cooperative Intrusion Detection Framework. In *Proceedings of the IEEE Symposium on Security and Privacy*, 2002.

[5] H. Debar and A. Wespi. Aggregation and Correlation of Intrusion-Detection Alerts. In *Proceedings of the International Symposium on Recent Advances in Intrusion Detection*, 2001.

[6] M. R. Endsley. Towards a Theory of Situation Awareness in Dynamic Systems. *Human Factors*, 37:32, 1995.

[7] Mica R. Endsley. Design and Evaluation for Situation Awareness Enhancement. In *Proceedings of the Human Factors Society 32nd Annual Meeting*, volume 1 of *Aerospace Systems: Situation Awareness in Aircraft Systems*, pages 97–101, 1988.

[8] C.W. Geib and R.P. Goldman. Plan Recognition in Intrusion Detection Systems. In *DARPA Information Survivability Conference & Exposition (DISCEX)*, 2001.

[9] Wei Jiang, Zhi hong Tian, Hong li Zhang, and Xin fang Song. A Stochastic Game Theoretic Approach to Attack Prediction and Optimal Active Defense Strategy Decision. In *IEEE International Conference on Networking, Sensing and Control (ICNSC'08)* , pages 648 –653, april 2008.

[10] Gary Klein and Beth Crandall. Recognition-Primed Decision Strategies. Technical report ARI Research Note 96-36, United States Army Research Institute for the Behavioral and Social Sciences, April 1996. `http://handle.dtic.mil/100.2/ADA309570`.

[11] C. Kruegel, W. Robertson, and G. Vigna. Using Alert Verification to Identify Successful Intrusion Attempts. *Practice in Information Processing and Communication (PIK)*, 27(4):219 – 227, October – December 2004.

[12] C. Kruegel, F. Valeur, and G. Vigna. *Intrusion Detection and Correlation: Challenges and Solutions*, volume 14 of *Advances in Information Security*. Springer, 2005.

[13] P. Liu and L. Li. A Game Theoretic Approach to Attack Prediction. Technical report, Penn State Cyber Security Group, 2002.

[14] V. Mehta, C. Bartzis, H. Zhu, E. Clarke, and J. Wing. Ranking attack graphs. In *Recent Advances in Intrusion Detection*, 2006.

[15] Sanjeeb Nanda and Narsingh Deo. The Derivation and Use of a Scalable Model for Network Attack Identification and Path Prediction. *JNW*, 3(4):64–71, 2008.

[16] P. Ning, Y. Cui, , and D. Reeves. Analyzing Intensive Intrusion Alerts via Correlation. In *Proceedings of the International Symposium on the Recent Advances in Intrusion Detection*, 2002.

[17] P. Ning, Y. Cui, and D. Reeves. Constructing Attack Scenarios through Correlation of Intrusion Alerts. In *Proceedings of the ACM Conference on Computer and Communications Security*, 2002.

[18] Steven Noel and Sushil Jajodia. Understanding complex network attack graphs through clustered adjacency matrices. In *21st Annual Computer Security Applications Conference (ACSAC 2005)*, pages 160–169. IEEE Computer Society, 2005.

[19] V. Paxson. Bro: A system for detecting network intruders in real-time. In *Proceedings of the 7th USENIX Security Symposium*, 1998.

[20] P. Porras, M. Fong, and A. Valdes. A Mission-Impact-Based Approach to INFOSEC Alarm Correlation. In *Proceedings of the International Symposium on the Recent Advances in Intrusion Detection*, 2002.

[21] X. Qin and W. Lee. Statistical Causality Analysis of INFOSEC Alert Data. In *Proceedings of the Symposium on Recent Advances in Intrusion Detection (RAID)*, 2003.

[22] X. Qin and W. Lee. Attack Plan Recognition and Prediction Using Causal Networks. In *Proceedings of the 20th Annual Computer Security Applications Conference*, 2004.

[23] J. Rasmussen. Skills, Rules, and Knowledge; Signals, Signs and Symbols, and Other Distinctions in Humans Performance Models. *IEEE Transactions on Systems, Man and Cybernetics*, 13:257, 1983.

[24] M. Roesch. Snort - Lightweight Intrusion Detection for Networks. In *Proceedings of the 13th Large Installation System Administration (LISA) Conference*, 1999.

[25] J.J. Salerno, M.L. Hinman, and D.M. Boulware. A situation awareness model applied to multiple domains. In *Proceedings of SPIE*, volume 5813, pages 65–74, 2005.

[26] G. Tadda, J.J. Salerno, D. Boulware, M. Hinman, and S. Gorton. Realizing situation awareness within a cyber environment. In *Multisensor, Multisource Information Fusion: Architectures, Algorithms, and Applications 2006*, volume 6242. SPIE, 2006.

[27] F. Valeur, G. Vigna, C. Kruegel, and R.A. Kemmerer. A Comprehensive Approach to Intrusion Detection Alert Correlation. *IEEE Transactions on Dependable and Secure Computing*, 1:146–169, 2004.

[28] L. Wang, A. Liu, and S. Jajodia. Using attack graphs for correlating, hypothesizing, and predicting intrusion alerts. *Computer Communications*, 29(15):2917–2933, 2006.

From Prey To Hunter[*]

Transforming Legacy Embedded Devices Into Exploitation Sensor Grids

Ang Cui
Department of Computer Science
Columbia University
New York NY, 10027, USA
ang@columbia.edu

Jatin Kataria
Department of Computer Science
Columbia University
New York NY, 10027, USA
jk3319@columbia.edu

Salvatore J. Stofo
Department of Computer Science
Columbia University
New York NY, 10027, USA
sal@columbia.edu

ABSTRACT

Our global communication infrastructures are powered by large numbers of legacy embedded devices. Recent advances in offensive technologies targeting embedded systems have shown that the stealthy exploitation of high-value embedded devices such as router and firewalls is indeed feasible. However, little to no host-based defensive technology is available to monitor and protect these devices, leaving large numbers of critical devices defenseless against exploitation. We devised a method of augmenting legacy embedded devices, like Cisco routers, with host-based defenses in order to create a stealthy, embedded sensor-grid capable of monitoring and capturing real-world attacks against the devices which constitute the bulk of the Internet substrate. Using a software mechanism which we call the Symbiote, a white-list based code modification detector is automatically injected *in situ* into Cisco IOS, producing a fully functional router firmware capable of detecting and capturing successful attacks against itself for analysis. Using the Symbiote-protected router as the main component, we designed a sensor system which requires no modification to existing hardware, fully preserves the functionality of the original firmware, and detects unauthorized modification of memory within 450 ms. We believe that it is feasible to use the techniques described in this paper to inject monitoring and defensive capability into existing routers to create an *early attack warning system* to protect the Internet substrate.

1. INTRODUCTION

The Internet is a dynamically changing network of many different kinds of devices, predominantly general purpose hosts and servers connected by a large collection of specialized embedded devices. Embedded devices such as routers, switches and firewalls constitutes the Internet's communication substrate. Devices such as VoIP, IPTV, power management and physical access control units provide a myriad of other specialized services. Most host-based security technologies deployed today are designed primarily to protect general purpose servers and hosts, leaving vast numbers of embedded devices, the Internet substrate itself, undefended against exploitation.

We present a new embedded device defense system designed make the internet substrate a safer environment. We believe it is technically feasible to inject security functionality *in situ* into legacy embedded systems to:

1. Provide security features to protect these devices against exploitation and rootkitting.
2. Create a large scale sensor grid providing new detection capability to identify attacks against embedded devices that are currently unmonitored.

Recent studies suggest that large populations of vulnerable embedded devices on the Internet are ripe for exploitation [8]. However, examples of successful exploits against such devices are rarely observed in the wild, despite the availability of proof-of-concept malware, known vulnerabilities and high monetization potential. We posit that our inability to monitor embedded devices for malware installation is a factor in this phenomenon. When deployed throughout the Internet substrate, the sensor system discussed in this paper will provide visibility into black-box embedded devices, allowing us to capture and analyze exploitation of embedded devices in real-time.

As a first step to show feasibility, we demonstrate a general method of transforming existing legacy embedded devices into exploitation detection sensors. We use Cisco firmware and hardware as the main demonstrative platform in this paper. However, the techniques described are not specific to any particular operating system or vendor, and can be directly applied to many other types of embedded devices.

In order to detect and capture successful attacks against Cisco routers for analysis, we engineered a system which automatically injects generic whitelist-based anti-rootkit functionality into standard IOS firmwares. Once injected, the augmented router firmware can be loaded onto physical Cisco routers, essentially transforming such devices into highly interactive router honeypots. As Section 8 shows, the resulting devices are fully functional, and can be deployed into production environments.

[*]Please note that Figures 1 and 2, along with portions of Section 5 is taken from a companion paper [9], and are present here so that our exposition has the appropriate background and completeness.

ACSAC '10 Dec. 6-10, 2010, Austin, Texas USA
Copyright 2010 ACM 978-1-4503-0133-6/10/12 ...$10.00.

The main challenge of creating an embedded device honeypot rests with the difficulties of injecting arbitrary detection code into proprietary, close-source, embedded devices with complex and undocumented operating systems. In order to overcome this challenge, we've created a software constructed called the Symbiote [9]. As Section 5 illustrates, the Symbiote, along with its payload, is injected *in situ* into an arbitrary host binary, in this case, Cisco IOS. The injection is achieved through a generic process which is agnostic to the operating environment of the host program. Figure 1 shows how a Symbiote is typically injected into a host program. In general, Symbiotes can inject arbitrary host-based defenses into black-box embedded device firmwares. For a full discussion of Symbiotes, please see [9]

The unique capabilities of the Symbiote construct allows us to overcome the complexities of injecting generic exploitation detection code into what is essentially an unknown black-box device. The original functionality of resulting Symbiote-injected embedded device firmware remains unchanged. A portion of the router's computational resources is diverted to a proof of concept Symbiote payload, which continuously monitors for unauthorized modifications to any static areas within the router's memory address space, a key side-effect of rootkit installation. As we demonstrate in Section 9, the portion of the CPU diverted to the Symbiote's payload is a configurable parameter, and directly effects the performance of the Symbiote payload; in this case, the detection latency of any unauthorized modification.

A monitoring system is constructed around the main component of our system, the Symbiote-injected IOS image. The Symbiote within the IOS firmware simultaneously performs checksums on all protected regions of the router's memory while periodically communicating with an external monitor via a covert channel. In the event of an unauthorized memory modification within the router, the Symbiote will raise an alarm to the external monitor, which then triggers the capture and analysis component of our system.

As Section 8 discusses, our monitoring system can be deployed in one of three ways; native deployment, emulated deployment, and shadow deployment. Due to the unique limitations of each deployment scenario, the capture and analysis mechanisms differ slightly. For example, when the Symbiote-injected firmware image is loaded into a physical Cisco router (native deployment), IOS's own core dump mechanism is used to capture the router's runtime state for analysis. This is less than ideal because, due to the hardware constraint of the Cisco device, we can not guarantee that the memory capture is performed atomically. Furthermore, since the core dump is generated by IOS's own (potentially compromised) code, the integrity of the output can not be fully trusted. In contrast, when the Symbiote-injected firmware is executed within Dynamips, a Cisco router emulator, on a general purpose computer (emulated deployment), the external monitor triggers a response which halts emulation of the compromised IOS image before initiating a full memory dump using the general purpose host computer. Thus, emulated deployment of our sensor can guarantee that the capture and analysis process can be done atomically without relying on potentially compromised code. Section 8 discusses the tradeoffs and advantages of all three deployments in detail.

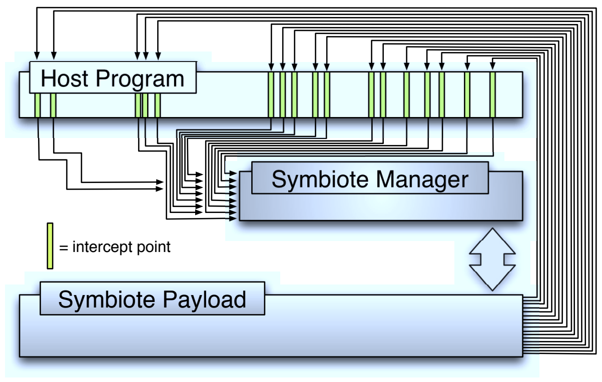


Figure 1: Logical overview of SEM injected into embedded device firmware. SEM maintains control of CPU by using large-scale randomized control-flow interception. SEM payload executes alongside original OS.

2. MOTIVATION

Several recent studies demonstrate that there are vast numbers of unsecured, vulnerable embedded devices on the internet [8], such devices are vulnerable to the same types of attacks as general purpose computers [3, 12], and can be systematically exploited in much the same way [1, 3, 5]. For example, various exploitable vulnerabilities [15, 13] and rootkits [16] have been found and disclosed for Cisco's router operating system, IOS. Cisco devices running IOS constitutes a significant portion of our global communication infrastructure, and are deployed within critical areas of our residential, commercial, financial, government, military and backbone networks.

Typical of the embedded security landscape, IOS is an aging system which does not employ standard protection schemes found within modern operating systems [16], and does not have any host-based anti-virus to speak of. In fact, not only is the installation of third-party anti-virus (which does not yet exist for IOS) not possible via any published OS interface, any attempt to do so will also violate the vendor's EULA and thus void existing support contracts.

Consider the availability of proof-of-concept exploits and rootkits, the wide gamut of high-value targets which can be compromised by the exploitation of devices like routers and firewalls, and the lack of host-based defenses within close-source embedded device firmwares. Such conditions should make the vast numbers of vulnerable embedded devices on the Internet highly attractive targets. Indeed, we have observed successful attempts to create botnets using Linux-based home routers [4]. As Section 4 shows, the necessary techniques of exploiting Cisco IOS and installing root-kits on running Cisco routers are well understood. The works presented within academic and blackhat circles, combined with anecdotal evidence of the systematic exploitation of embedded network devices within the last decade suggests that real-world exploitation of Cisco routers is not only possible, but likely an undetected reality.

Documented cases of embedded device exploitation are still relatively rare. High-value embedded targets like enterprise networking equipment have seemingly eluded exploitation. It is possible that the exploitation of devices like Cisco routers is still beyond the technical capabilities of the blackhat community. However, it is far more plausible that stealthy, targeted attacks against high-value embedded devices have eluded detection due to our inability to gain visibility into the internals of such devices. It is quite possible routers have been successfully attacked and are compromised without anyone's knowledge except the UE sellers who offer them for sale.

Whether or not stealthy exploitation of embedded devices is a reality today, we can confidently anticipate that attacks against such defenseless, high-value targets is inevitable. Therefore, analysis and mitigation of embedded device exploitation is crucial to the integrity of the Internet substrate. We believe that accurate, real-time detection of such attacks is an important first step towards understanding the realities of the embedded security threat. Furthermore, we believe the ability to inject host-based security into existing legacy devices will be instrumental in mounting a realistic defense of existing embedded devices.

The Symbiote structure presented in this paper is designed specifically to abstract away the technical challenges of injecting third-party security into a diverse range of embedded devices. This device agnostic foundation allows us to look beyond specific hardware and firmware in order to create a general body of embedded defense methodologies which can be feasibly applied to all existing devices.

3. THREAT MODEL

We are interested in detecting, capturing and analyzing successful injection of rootkits into IOS at runtime. We assume that the attacker is technically sophisticated and has access to both zero-day vulnerabilities as well as a reliable rootkit which persistently alters the behavior of the victim device's OS, yielding covert root access to the attacker. We assume that the injected rootkit will patch specific portions of the router's code in order to create a hidden backdoor for the attacker. In other words, we assume that the rootkit will alter regions of memory within the router that is meant to be **static** during normal execution. Static sections within IOS firmware image typically include the .txt, .rodata, .firmware, .sdata and large portions of the .data sections. Furthermore, the boot-loader (rommon) section of the router, as well as all associated configuration files (startup configuration, running configuration, etc) can also be monitored for unauthorized modification.

While our current threat model encompasses all published IOS rootkitting techniques to date, it is probable that a covert backdoor can be created within an IOS router without modifying static regions of the router's memory. The proposed detection payload will not detect exploits which leave no persistent change within the victim device. However, the Symbiote-based injection scheme described in this paper can be extended to monitor for anomalies within dynamic sections of the target device, extending our whitelist-based detector into a full-blown host-based anomaly detector (See Section 10).

Furthermore, it is possible for sophisticated attacks to attempt to disable the Symbiote prior to the actual exploitation of the victim device. Since the Symbiote structure described in this paper is a software-based defense, absolute integrity of the Symbiote cannot be guaranteed. In the general case, Symbiotes can be fortified with the introduction of specialized hardware. However, such a solution is not feasible when considering the realm of legacy embedded devices. Instead, Section 6 illustrates a general method of increasing the computational complexity of a successful bypass of the Symbiote defense without relying on additional hardware.

4. RELATED WORK

Relatively little work has been done to detect and capture sophisticated attacks against embedded devices. However, such problems have been well studied for general purpose computers and operating systems. Numerous rootkit and malware detection and mitigation mechanisms have been proposed in the past but largely target general purpose computers. Commercial products from vendors like Symantec, Mcafee/Intel, Kapersky and Microsoft [2] all advertise some form of protection against kernel level rootkits. Kernel integrity validation and security posture assessment capability has been integrated into several Network Admission Control (NAC) systems. These commercial products largely depend on signature-based detection methods and can be subverted by well known methods [18, 19, 20]. Sophisticated detection and prevention strategies have been proposed by the research community. Virtualization-based strategies using hypervisors, VMM's and memory shadowing [17] have been applied to kernel-level rootkit detection. Others have proposed detection strategies using binary analysis [11], function hook monitoring [23] and hardware-assisted solutions to kernel integrity validation [22].

The above strategies may perform well within general purpose computers and well known operating systems but have not been adapted to operate within the unique characteristics and constraints of embedded device firmware. Effective prevention of binary exploitation of embedded devices requires a rethinking of detection strategies and deployment vehicles.

Our methodology transforms standard legacy embedded devices into exploitation detectors. This is similar to existing honeypot-based IDS strategies, which generally involves the use of intentionally vulnerable systems to log, capture and analyze attacks levied against it. Many honeypot-based systems have been proposed. Few focus on the use or protection of embedded devices. For example, Ghourabi *et al.* recently proposed the use of simulated honey routers to study protocol attacks against BGP [10].

In general, honeypots can be native, emulated or simulated, and can involve a single machine or a vast network of simulated nodes. Many off-the-shelf honeypot systems exist for general purpose computers. However, such systems are not without flaws. For example, simulated honeypots disguises themselves as vulnerable systems but does not expose any actual vulnerabilities to the attacker. Therefore, minimizing false-positives in such systems is a challenge. Furthermore, simulated honeypots may catch indiscriminate exploitation attempts, but will rarely fool sophisticated attackers in highly targeted attacks. Thus, native and emulated honeypots which exposes real vulnerabilities to the attacker are much better suited for detecting sophisticated, targeted attacks.

Guards, originally proposed by Chang and Atallah [6], is another technology which uses mechanisms of action similar to Symbiotes. A Guard is a simple piece of security code which is injected into the protected software using binary rewriting techniques similar to our Symbiote system. Once injected, a guard will perform tamper-resistance functionality like self-checksumming and software repair. However, Guards have no mechanism to pause and resume its computation, the entire Guard routine must complete execution each time it is invoked. This limits the sophistication of what each Guard can realistically perform, especially when Guards are used in time sensitive software and real-time embedded devices.

Devices like Cisco routers are black-box systems utilizing large numbers of undocumented proprietary hardware components. The injection of new code into proprietary firmware and the emulation of specialized and undocumented hardware makes the creation native and emulated honeypots for embedded devices challenging. As Section 5 describes, the unique capabilities of the Symbiote construct allows us to overcome the above challenges in order to transform standard Cisco IOS firmware and hardware into highly believable native router honeypots.

5. MEET SYMBIOTE

For a full discussion of Symbiotic Embedded Machines, please see [9]. The Symbiote is a software construct that is injected *in situ* into a host program to provide the following four fundamental security properties.

1. The Symbiote has full visibility into the code and execution state of its host program, and can either passively monitor or actively react to the observed events at runtime.
2. The Symbiote executes along side the host software. In order for the host to function as before, it's injected Symbiote must execute, and vice versa.
3. The Symbiote is an autonomous entity which is hardened to defend against unauthorized modification or removal once it is injected into the host program.
4. No two instantiations of the same Symbiote are the same. Each time a Symbiote is created, its code is randomized and mutated, rendering signature based detection methods and attacks requiring predictable memory and code structures within the Symbiote ineffective.

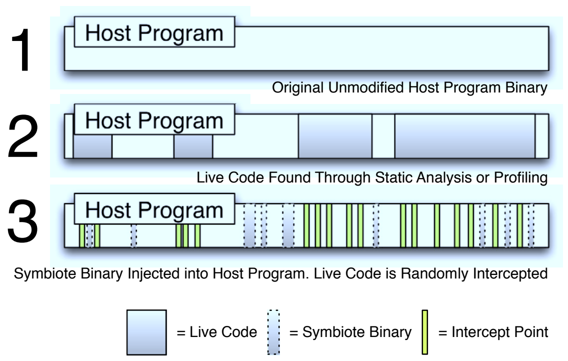

1
Host Program
Original Unmodified Host Program Binary
2
Host Program
Live Code Found Through Static Analysis or Profiling
3
Host Program
Symbiote Binary Injected into Host Program. Live Code is Randomly Intercepted
= Live Code
= Symbiote Binary
= Intercept Point

Figure 2: Symbiote Injection Process.

Figure 1 shows the three logical components of Symbiotes: Control-Flow Interceptors, Symbiotic Embedded Machine Manager (SEMM) and the Symbiote Payload. Together, all three components are injected *in situ* into the target embedded device firmware. Since the Symbiote is injected *in situ*, the size of the resulting firmware image is unchanged. For example, the current implementation of the Symbiote Manager, along with the rootkit detection payload requires only approximately 1600 bytes to be injected into IOS.

Figure 2 illustrates the three step Symbiote injection process. First, analysis is performed on the original host program in order to determine areas of live code, or code that will be run with high probability at runtime. Second, intercept points are chosen randomly from the host program. Lastly, the Symbiote Manager, Symbiote payload and a large number of control-flow intercepts are injected into the host program binary, yielding a Symbiote protected host program.

The Symbiote randomly intercepts a large number of functions as a means to divert periodically and consistently a small portion of the device's CPU cycles to execute its payload. This approach allows the Symbiote to remain agnostic to operating system specifics while executing its payload alongside the original OS. The Symbiote payload has full access to the internals of the original OS but is not constrained by it. This allows the payload to carry out functionality which might not be possible under the original OS. In the case of Cisco IOS for example, a process watchdog timer will forcibly terminate any process which executes for more than several seconds. However, since the Symbiote payload executes in time-slices randomly distributed throughout many unrelated processes, the Symbiote payload can execute indefinitely, circumventing the watchdog timer entirely.

Stealth is a byproduct of the SEM structure. In the case of IOS, no diagnostic tool available within the OS (short of a full memory dump) can detect the presence of the SEM payload because it manipulates no OS specific structure and is effectively invisible to the OS. The impact of the SEM payload is further hidden by the fact that CPU utilization of the payload is not reported within any single process under IOS and is distributed randomly across a large number of unrelated processes.

Once the Symbiote Manager gains control of the CPU, it allocates a certain number of cycles for the execution of its Symbiote payload (in this case, a checksuming mechanism). After the payload completes its execution burst, control of the CPU is returned to the Symbiote Manager, which in turn resumes the execution of the original host program.

The Symbiote Manager acts as a job scheduler, treating the entire host program as one process, and its Symbiote payload as the other. Traditional scheduling strategies can be used to determine the proper CPU resource distribution between the Symbiote and its host program. In general, this involves the optimization of both the *frequency* of context switches as well as the *duration* of the Symbiote payload's execution bursts.

The proposed Symbiote payload detects unauthorized code modification through the computation of checksums over static regions of memory. Therefore, a delay exists between the time of the code modification and its detection. In general we refer to the time between the occurrence of an unauthorized event and its detection as the *detection latency.* Intuitively, the amount of CPU resources diverted to the Symbiote payload should be inversely proportional to the detection latency, and thus directly proportional to the performance of our detector. In the case of Cisco IOS, and many other embedded systems, an over allocation of CPU resources to the Symbiote can adversely affect the performance of the protected host device. In practice, we have found that it is beneficial to frequently interleave the host program's execution with short Symbiote payload execution bursts. This allows the Symbiote payload to compute at acceptable rates while minimizing the impact on the real-time nature of Cisco routers.

The Symbiote scheduling problem is arguably simple as it involves only two "tasks". However, performing such a task safely in an OS agnostic manner on embedded systems presents several interesting complexities. A full discussion of potential Symbiote scheduling algorithms is out of the scope of this paper. However, in the case of our IOS exploitation detection Symbiote, the performance and overhead characteristics of several scheduling strategies are discussed in Section 9.

6. SELF-MONITORING SYMBIOTES

We must consider ways to protect the Symbiote itself against attack and removal. The polymorphic nature of the Symbiote and its payload makes signature-based attacks against it ineffective. To further raise the bar, multiple Symbiotes within a protected host program can be configured in a self-monitoring monitor arrangement. As proposed by Stolfo, Greenbaum and Sethumadhavan [21], a network of monitors can be constructed, such that an alarm will be raised if any subset of monitors are compromised or deactivated, or if any critical condition monitored by the system is violated. Consider Figure 3, which shows three independent Symbiotes arranged in a full-mesh monitoring network. In this arrangement, each Symbiote monitors a specific critical condition, i.e., the output of their Symbiote Payload, while simultaneously monitoring the operational status of the other two Symbiotes within the network. If one or more of the Symbiotes are corrupted or disabled, the remaining Symbiotes within the network will raise an alarm. Similarly, if all three Symbiotes are simultaneously deactivated, an external sensor can also detect this event and raise an alarm. Note that the three Symbiotes shown in Figure 3 need not be located within the same host router. Large networks of embedded device sensors can be collectively protected in this mutually defensive arrangement. Using Symbiotes in this fashion is a topic of ongoing research.

7. EXPLOITATION DETECTOR

IOS rootkit and malware code is generally not publicly available. However, a survey of published persistent rootkit techniques reveals a commonality in their *modus operandi.* Specifically, rootkits such as [16, 13, 7] all modify some region of static IOS memory in order to inject their rootkit payload into the victim router. Thus, we implemented a white-list strategy to detect IOS malcode and rootkits described previously.

Known rootkits operate by hooking into and altering key functions within IOS. To do this, specific binary patches must be made to executable code. Therefore, a continuous integrity check on all **static areas** of Cisco IOS will detect all function hooking and patching attempts made by rootkits and malware. The rootkit detection payload described below is not specific to IOS, and can be used on other embedded operating systems as well. As Section 9 shows, our Symbiote payload accurately detects unauthorized modification of any monitored region of memory within milliseconds, and will accurately detect [16, 13, 7] immediately after successful exploitation of the victim device.

In the case of Cisco IOS, several large contiguous segments of the router's memory address space can be monitored using the checksumming mechanism described above. Figure 4 illustrates the memory layout of a typical IOS firmware image on a Cisco router. The darkened regions represent areas of the router's firmware which can be safely monitored by our checksumming mechanism. For example, regions containing executable code (text and firmware), and static data (rodata, ctors, sdata sections) should clearly not be modified at runtime. In practice, the typical IOS firmware contains large contiguous sections of memory which should semantically remain static during the normal operation of the router.

8. DESIGN AND OPERATION

Our sensor system has three components; a Symbiote-protected router, a monitoring station, and a capture and analysis system which automatically collects and analyzes forensics data once an alarm is triggered. The Symbiote within the IOS firmware simultaneously performs checksums on all protected regions of the router's memory while periodically communicating with an external monitor via a covert channel. In the event of an unauthorized memory modification within the router, the Symbiote will raise an alarm to the monitor, which then triggers the capture and analysis component of our system.

The proposed exploitation detection sensor can be deployed in one of at least three ways; natively, emulated within a general purpose computer, or as a shadow replica for a production device. The implementation of the monitoring station and capture and analysis engine changes depending on how the Symbiote-injected router firmware is executed; natively on embedded hardware or emulated on a general purpose computer.

When deployed natively, the monitor and capture components are integrated into the Symbiote payload and injected directly into Cisco hardware, producing a standalone sensor. When the detection payload raises an alarm, the Symbiote immediately triggers the core dump functionality from within IOS. This causes the bulk of the router's execution state to be captured and transferred via FTP or TFTP.

When deployed as an emulated sensor, using Dynamips for example, the monitoring and capture components of the sensor are implemented within the emulator. This reduces the footprint of the Symbiote and allows us to perform more sophisticated capture and analysis on the server running the emulation. For example, Dynamips was modified to continuously monitor a region of the router's memory for an encoded marker, which is set by the Symbiote payload only when an alarm is raised.

For testing purposes, we chose to modify a portion of the text that is printed when the "show version" command is invoked. In practice, many better covert channels can be used to communicate between the Symbiote and the router emulator.

In order to transform large populations of embedded devices into massive embedded exploitation sensor-grids, the native deployment is the most efficient and practical. For the purposes of testing and validation of our approach, the emulated deployment scenario is most appropriate. The shadow

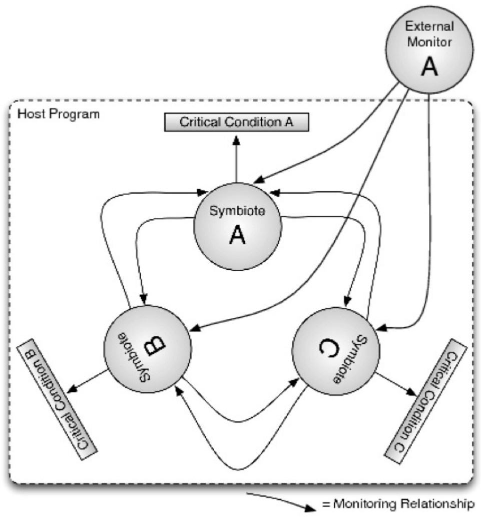


Figure 3: Full Mesh Self-Monitoring Symbiote Network

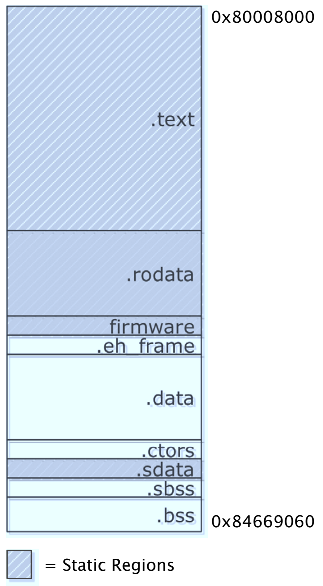


Figure 4: Memory layout of a typical Cisco IOS router

deployment is best for capturing and analyzing IOS exploits in mission critical production environments.

8.1 Native Sensor Deployment

In the native deployment scenarios, the Symbiote-injected firmware is loaded directly onto the target embedded device, i.e. a Cisco router. The Symbiote payload executes natively on the embedded hardware, alongside the original firmware. Native deployment allows the Symbiote to operate in embedded systems for which emulation is not feasible. For example, a large portion of Cisco devices can not be emulated by existing software due to the use of undocumented, proprietary hardware. In practice, most modern high-performance networking equipment falls within this category. Therefore, native deployment is most practical for injecting Symbiotic defenses into the diverse range of embedded devices found on the Internet substrate.

8.2 Emulated Sensor Deployment

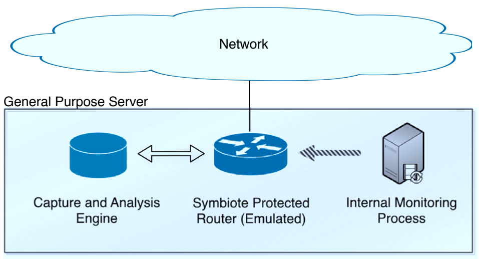

Network
General Purpose Server
Capture and Analysis Engine
Symbiote Protected Router (Emulated)
Internal Monitoring Process

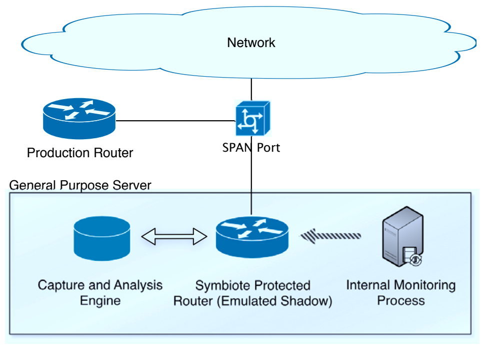
Figure 5: Emulated Deployment of Symbiote-based Cisco IOS Detector

Figure 5 illustrates a typical *emulated* deployment of our sensor. Instead of running the Symbiote-injected firmware natively on embedded hardware, the firmware is emulated on a general purpose computer. In the case of Cisco IOS, Dynamips is used to emulate specific devices such as 7200 series routers used in our testing and evaluation environment. This differs from *simulated* honeypots in a significant way: the use of actual IOS firmware. The emulation of real Cisco IOS allows us to create highly interactive honeypots which exposes real IOS vulnerabilities to potential attackers.

The emulated deployment has several advantages which make it the ideal approach for developing and testing experimental prototype Symbiotes. First, debugging proof of concept Symbiotes in an emulated environment is slightly more convenient than doing so on native embedded hardware. Second, the general purpose computer which hosts the emulation usually has far greater computational capacity than the embedded hardware which it is emulating. Therefore, computation can be offloaded from the Symbiote payload onto the general purpose host computer. This can potentially allow the Symbiote payload to perform complex computations not feasible on actual embedded hardware. The Symbiote payload presented in this paper is simple and requires relatively little CPU power. However, this can be replaced with payloads more akin to behavior based anomaly detectors which can require significantly more resources. The development of Symbiote-based anomaly detection mechanisms is an area of active research (See Section 10).

Lastly, the emulated sensor deployment can usually simplify the capture and analysis component of the sensor. In the case of the sensor presented in this paper, we modified the Dynamips emulator to *atomically* capture the entire memory state of the IOS router once the Symbiote payload emits an alarm. The Dynamips emulator conveniently allows us to halt the router's CPU briefly while the memory capture takes place on the host computer. Once this operation completes, the memory snapshot, along with all network traffic received by the router is automatically processed and archived for analysis. Once the Symbiote payload emits an alarm, our modified Dynamips emulator continuously dumps the memory state of the router at a configurable frequency.

8.3 Shadow Sensor Deployment



Network
Production Router
SPAN Port
General Purpose Server
Capture and Analysis Engine
Symbiote Protected Router (Emulated Shadow)
Internal Monitoring Process

Figure 6: Shadow Deployment of Symbiote-based Cisco IOS Detector

In order to detect exploitation against high-performance embedded devices within production environments, we must be able to deploy Symbiote-based sensors in a way which will not cause unintentional service outages on the monitored devices. In such cases, the use of a second, identical embedded device as a *shadow* sensor is most appropriate. Figure 6 illustrates a typical shadow sensor deployment.

As the name suggests, incoming network traffic is mirrored from the production embedded device to a Symbiote-injected shadow device, which runs the same firmware as the production device. The Symbiote sensor injected into the shadow device continuously monitors the shadow device, quietly emitting alerts when malicious activity is detected.

The performance of the shadow sensor is critical, as it must be able to keep up with the production router. Thus, minimizing the control-plane latency and computational overhead of the Symbiote is critical to the effectiveness of the detection system. We discuss preliminary performance data in the next section. The development of Symbiote-based shadow sensors is an area of active research.

9. PERFORMANCE AND OVERHEAD

We measure the performance and overhead of our Symbiote-based exploit detector using two quantitative metrics: *computational overhead* and *detection latency*. The Symbiote-protected router is an emulated Cisco 7200 series router running IOS 12.3. Two neighbor routers are used to verify that the Symbiote-protected router's original functionality is unchanged. One neighbor router is an emulated 7200 series router running standard IOS 12.3. The other neighbor router is a physical Cisco 2921 router running IOS 12.5. Each router is configured to expose a cross-section of functionality typically seen on production routers. Specifically, a large number of local loopback interfaces are configured on each router. OSPF routing is enabled on all three routers, along with a suite of standard services like IP-SLA, SNMP, HTTP{S} and SSH.

A stress-test script automatically generates network traffic throughout the test environment, and periodically accesses services on all the test routers. All routers are continuously monitored to ensure that all services operate correctly throughout testing. The workload script also periodically forces route-table re-calculations by perturbing the various OSPF routers on the network. In effect, the stress-test script simulates a typical use profile for the IOS routers in the test environment. The same stress-test script is run against several variants of the Symbiote-injected IOS firmware in order to illustrate key performance features of our system.

The computational overhead and performance of our system is a configurable parameter. As the figures in this section shows, the scheduling algorithm used within the Symbiote Manager directly impacts the resource consumption of the Symbiote payload, and thus the overall utilization of the host device as well as the detection latency. Two scheduling algorithms are discussed in this section: *fixed burst-rate* and *inverse-adaptive*.

As the name suggests, the fixed burst-rate scheduling algorithm instructs the Symbiote payload to execute for a fixed burst-rate each time the Symbiote Manager is invoked through a randomly placed execution intercept. On the other hand, the inverse-adaptive scheduling algorithm calculates the payload burst-rate based on the elapsed time since the Symbiote Manager was last invoked; the longer the elapsed time, the longer the burst-rate.

Intuitively, we can expect the fixed burst-rate scheduling algorithm to execute the Symbiote payload *more frequently* as the host system becomes more utilized. This simple algorithm executes the Symbiote payload more frequently when the Cisco router is heavily utilized, and less frequently when the router is idle. In contrast, the inverse-adaptive scheduling algorithm increases Symbiote payload burst-rate when the system is under-utilized, and throttles back the Symbiote payload when the router is under high load.

We analyze the performance of 15 Symbiote-injected IOS images under the same stress-test; 7 variants using the fixed burst-rate Symbiote scheduler and 8 variants using the inverse-adaptive Symbiote scheduler. As the next three subsections show, the fixed burst-rate Symbiote scheduler aggressively executes the Symbiote payload, and achieves the least detection latency (approximately 400 ms). However, this aggressive scheduler tends to amplify CPU utilization of the protected router, causing very high control-plane latency when the router is under load. Although the higher fixed burst-rate values like 0x7FF and 0xFFF detected IOS modification very quickly, it also caused the router's control-plane to be less responsive.

In contrast, the inverse-adaptive Symbiote scheduler produced slightly longer detection latencies (approximately 450 ms), but was able to significantly reduce the control-plane latency of the host router, even under high load.

9.1 Computational Overhead

The same stress-test script is run against various versions of the Symbiote-injected IOS image in order to show how the Symbiote Manager's scheduling algorithm affects the CPU utilization of the router. Two major scheduling algorithms are measured: fixed burst-rate (Figure 7) and inverse-adaptive (Figure 8). **Burst-rate** values presented in the next two sections represent the number of iterations of the main Symbiote payload executed each time the Symbiote Manager is invoked.

Figure 7 shows the CPU utilization of 7 variants of the *fixed burst-rate* Symbiote scheduler, which unconditionally executes the Symbiote payload for a constant number of CPU cycles each time the Symbiote is invoked via its many control-flow intercepts. The units used, burst-rate, is the number of iterations of the checksum Symbiote payload that is executed each time the Symbiote Manager is invoked.

This Symbiote scheduler disregards the current CPU utilization of the host device. At higher burst-rate values like 0x7FF and 0xFFF, the router's CPU utilization tends to remain above 95% under heavy load, causing large spikes in control-plane latency. (See Figure 11)

Figure 8 shows the CPU utilization of 8 variants of the *inverse-adaptive* Symbiote scheduler, compared with the baseline CPU utilization of the unmodified IOS image under the same stress-test. The inverse-adaptive scheduler is configured with *maximum* burst-rates from 0x1FFFF to 0xFFFFFF. Unlike the fixed burst-rate Symbiote scheduler, the inverse-adaptive scheduler throttles how much the CPU is diverted to the Symbiote based on current host device utilization. As a result, Symbiotes with inverse-adaptive schedulers can achieve comparable detection latencies while significantly reducing its impact on the host router's control-plane latency. (Compare Figure 11 and Figure 12).

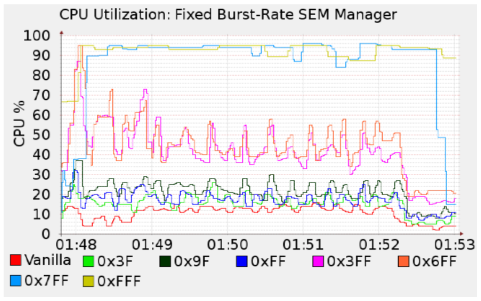


Figure 7: CPU Utilization: Fixed Burst-Rate SEM Manager

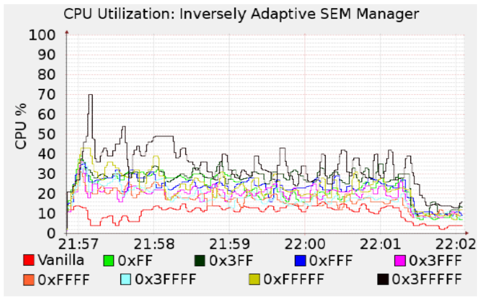


Figure 8: CPU Utilization: Inverse-Adaptive SEM Manager

9.2 Detection Performance

In order to measure the detection latency of our exploitation detection Symbiote, a simple vulnerability which allows arbitrary memory modification is artificially introduced into the Symbiote-injected IOS image. Using an automated script, this vulnerability is triggered, modifying a random byte within monitored memory regions. A timer is simultaneously started in order to measure the time it takes the Symbiote payload to detect the event.

Figure 9 shows a roughly linear relationship between the Symbiote's fixed burst-rate value and the Symbiote's detection latency. As expected, the Symbiote detection latency decreases as the Symbiote payload's execution burst-rate increases. However, as Figure 11 shows, the fixed burst-rate Symbiote scheduler causes significant increases in the router's control-plane latency.

Figure 10 shows the detection latency of Symbiotes using the inverse-adaptive scheduler. As the figure shows, these Symbiotes can achieve comparable detection latency values as the fixed burst-rate versions, but as Figure 12 shows, the Symbiote's impact on the router's control-plane is significantly reduced.

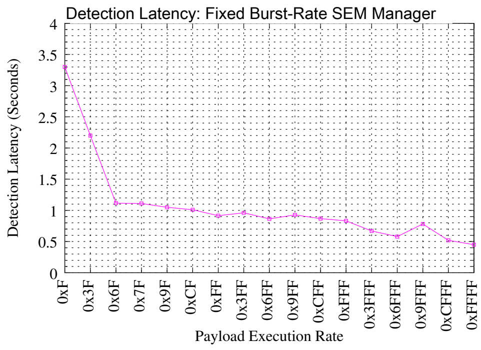


Figure 9: Detection Latency: Fixed Burst-Rate SEM Manager

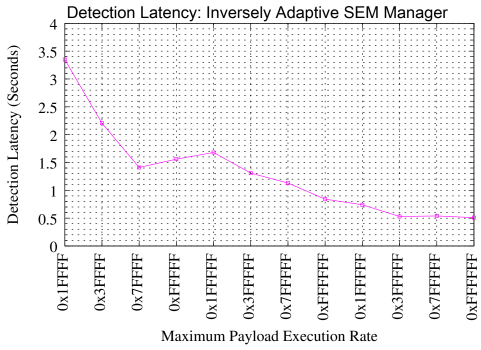


Figure 10: Detection Latency: Inverse-Adaptive SEM Manager

9.3 Control-Plane Latency

Control-plane latency is an indicator of how responsive the router is. High control-plane latency can cause a router to drop routing adjacencies and break various time-sensitive network protocols. Note, however, that this measurement will not significantly affect the latency of traffic passing through the router, as most modern routers have hardware-accelerated forwarding engines which are decoupled from the control-plane.

Control-plane latency is measured by sending ICMP-echo messages from the test PC to the router's local loopback interface. The round-trip-time is collected and shown in Figure 11 for Symbiotes using fixed burst-rate scheduler variants, and in Figure 12 for Symbiotes using inverse-adaptive scheduler variants. Clearly, the inverse-adaptive Symbiote scheduler significantly reduces the Symbiote's impact on the host router's control-plane latency while achieving comparable detection latency values as fixed burst-rate Symbiotes.

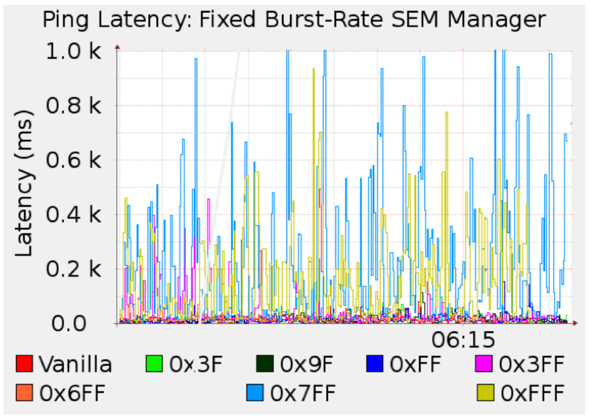


Figure 11: Ping Latency: Fixed Burst-Rate SEM Manager

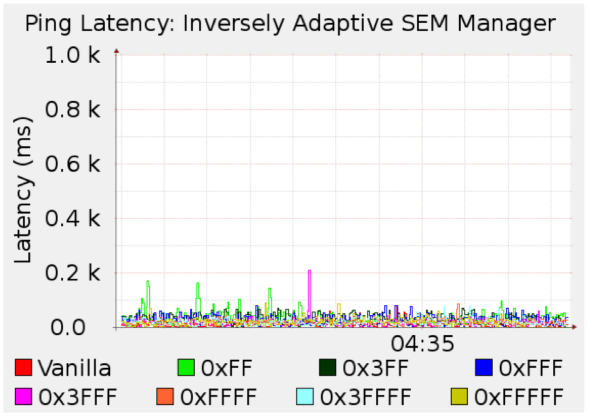


Figure 12: Ping Latency: Inverse-Adaptive SEM Manager

9.4 Discussion

Preliminary performance results shown in this section suggests that high performance exploitation detection is possible in Cisco IOS. Furthermore, an optimized Symbiote scheduling algorithm can greatly improve performance of the overall sensor system by reducing both detection latency and the Symbiote's impact on the router's control-plane latency. Optimization of the detection latency and the induced control-plane latency is an area of active research.

10. FUTURE WORK

The Symbiote-based sensor presented in this paper is a first step towards demonstrating the feasibility and novel capability of Symbiotic defense systems. The Symbiote structure allows complex payloads to be injected into legacy embedded devices, allowing the payload to safely execute alongside the original firmware without altering the embedded device's functionality. The checksumming payload we injected into Cisco IOS can be replaced with a wide range of defensive payloads. Below are several new Symbiote payloads currently under development.

10.1 Embedded Self-Healing

The checksumming Symbiote payload discussed in this paper can be extended to **reverse** unauthorized modification of memory after it is detected. A self-healing Symbiote payload can be used to identify and restore regions of memory which have been maliciously modified.

10.2 Embedded Anomaly Detector

Symbiote payloads can implement existing anomaly detection algorithms. For example, behavior modeling strategies which monitor resource utilization, control and data flow patterns can be injected into embedded devices via Symbiote payloads.

10.3 Large-Scale Embedded Sensor Grid

The exploitation detection sensor described in this paper can be injected into large numbers of embedded devices like Cisco routers in order to monitor and analyze 0-day exploitation of embedded devices. We believe the use of Symbiote-based exploitation sensors in the wild is a feasible and effective way of monitoring and analyzing exploits levied against the internet substrate. A large-scale Symbiote-based sensor grid can potentially give us real-time visibility into embedded device exploitation on a global scale.

Furthermore, Symbiotes can be used to transform embedded devices into other kinds of sensor grids as well. Symbiotes can allow us to use hardware components of embedded devices in novel ways not intended by its original design. For example, many power-consuming, EM emitting components can be transformed into covert communication channels. Existing sensors on embedded devices, combined with such covert channels can transform a wide gamut of innocuous embedded devices into a web of remotely controlled mobile sensors.

11. CONCLUSION

The Symbiote mechanism can be used to augment legacy embedded devices with novel functionality in an OS agnostic manner. The applications of this capability are numerous, and will help make the introduction of modern host-based defenses on existing embedded devices a feasible reality. The checksumming Symbiote payload described in this paper is a starting point in demonstrating the unique advantages of Symbiotic defense systems. We have demonstrated that the

Symbiote can automatically augment Cisco IOS with effective anti-rootkitting capabilities. This accomplishment has laid the foundation for the construction of a large sensor-grid of legacy embedded devices in order to accurately detect and analyze the exploitation of the devices which make up the fabrics of our global communication infrastructures.

12. ACKNOWLEDGEMENTS

This material is based on research sponsored by Air Force Research labs under agreement number FA8750-09-1-0075. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon. This material is also based on research sponsored by DARPA contract: CRASH program, SPARCHS, FA8750-10-2-0253.

13. REFERENCES

[1] kaiten.c IRC DDOS Bot. http://packetstormsecurity.nl/irc/kaiten.c.

[2] Microsoft Corporation, Kernel Patch Protection: Frequently Asked Questions. http://tinyurl.com/y7pss5y, 2006.

[3] The End of Your Internet: Malware for Home Routers, 2008. http://tinyurl.com/3d9yv9l.

[4] Network Bluepill. Dronebl.org, 2008. http://www.dronebl.org/blog/8.

[5] New worm can infect home modem/routers. APCMAG.com, 2009. http://apcmag.com/Content.aspx?id=3687.

[6] Hoi Chang and Mikhail J. Atallah. Protecting software code by guards. In Tomas Sander, editor, *Digital Rights Management Workshop*, volume 2320 of *Lecture Notes in Computer Science*, pages 160–175. Springer, 2001.

[7] Ang Cui, Jatin Kataria, and Salvatore J. Stolfo. Killing the myth of cisco ios diversity: Towards reliable, large-scale exploitation of cisco ios, 2011. 5th USENIX Workshop on Offensive Technologies.

[8] Ang Cui and Salvatore J. Stolfo. A quantitative analysis of the insecurity of embedded network devices: results of a wide-area scan. In Carrie Gates, Michael Franz, and John P. McDermott, editors, *ACSAC*, pages 97–106. ACM, 2010.

[9] Ang Cui and Savaltore J. Stolfo. Defending legacy embedded devices with software symbiotes. In Robin Sommer, Davide Balzarotti, and Gregor Maier, editors, *RAID*, volume 6961 of *Lecture Notes in Computer Science*. Springer, 2011.

[10] Abdallah Ghourabi, Tarek Abbes, and Adel Bouhoula. Honeypot router for routing protocols protection. In Anas Abou El Kalam, Yves Deswarte, and Mahmoud Mostafa, editors, *CRiSIS*, pages 127–130. IEEE, 2009.

[11] Christopher Krügel, William K. Robertson, and Giovanni Vigna. Detecting kernel-level rootkits through binary analysis. In *ACSAC*, pages 91–100. IEEE Computer Society, 2004.

[12] Felix "FX" Linder. Cisco Vulnerabilities. In *In BlackHat USA*, 2003.

[13] Felix "FX" Linder. Cisco IOS Router Exploitation. In *In BlackHat USA*, 2009.

[14] Richard Lippmann, Engin Kirda, and Ari Trachtenberg, editors. *Recent Advances in Intrusion Detection, 11th International Symposium, RAID 2008, Cambridge, MA, USA, September 15-17, 2008. Proceedings*, volume 5230 of *Lecture Notes in Computer Science*. Springer, 2008.

[15] Michael Lynn. Cisco IOS Shellcode, 2005. In BlackHat USA.

[16] Sebastian Muniz. Killing the myth of Cisco IOS rootkits: DIK, 2008. In EUSecWest.

[17] Ryan Riley, Xuxian Jiang, and Dongyan Xu. Guest-transparent prevention of kernel rootkits with vmm-based memory shadowing. In Lippmann et al. [14], pages 1–20.

[18] Dror-John Roecher and Michael Thumann. NAC Attack. In *In BlackHat USA*, 2007.

[19] Skywing. Subverting PatchGuard Version 2, 2008. Uninformed,Volume 6.

[20] Yingbo Song, Pratap V. Prahbu, and Salvatore J. Stolfo. Smashing the stack with hydra: The many heads of advanced shellcode polymorphism. In *Defcon 17*, 2009.

[21] Salvatore J. Stolfo, Issac Greenbaum, and Simha Sethumadhavan. Self-monitoring monitors. Technical Report cucs-026-09, Columbia University Computer Science Department, April 2009.

[22] Vikas R. Vasisht and Hsien-Hsin S. Lee. Shark: Architectural support for autonomic protection against stealth by rootkit exploits. In *MICRO*, pages 106–116. IEEE Computer Society, 2008.

[23] Zhi Wang, Xuxian Jiang, Weidong Cui, and Xinyuan Wang. Countering persistent kernel rootkits through systematic hook discovery. In Lippmann et al. [14], pages 21–38.

BareBox: Efficient Malware Analysis on Bare-Metal

Dhilung Kirat
University of California, Santa Barbara
dhilung@cs.ucsb.edu

Giovanni Vigna
University of California, Santa Barbara
vigna@cs.ucsb.edu

Christopher Kruegel
University of California, Santa Barbara
chris@cs.ucsb.edu

ABSTRACT

Present-day malware analysis techniques use both virtualized and emulated environments to analyze malware. The reason is that such environments provide isolation and system restoring capabilities, which facilitate automated analysis of malware samples. However, there exists a class of malware, called VM-aware malware, which is capable of detecting such environments and then hide its malicious behavior to foil the analysis. Because of the artifacts introduced by virtualization or emulation layers, it has always been and will always be possible for malware to detect virtual environments.

The definitive way to observe the actual behavior of VM-aware malware is to execute them in a system running on real hardware, which is called a "bare-metal" system. However, after each analysis, the system must be restored back to the previous clean state. This is because running a malware program can leave the system in an instable/insecure state and/or interfere with the results of a subsequent analysis run. Most of the available state-of-the-art system restore solutions are based on disk restoring and require a system reboot. This results in a significant downtime between each analysis. Because of this limitation, efficient automation of malware analysis in bare-metal systems has been a challenge.

This paper presents the design, implementation, and evaluation of a malware analysis framework for bare-metal systems that is based on a fast and rebootless system restore technique. Live system restore is accomplished by restoring the entire physical memory of the analysis operating system from another, small operating system that runs outside of the target OS. By using this technique, we were able to perform a rebootless restore of a live Windows system, running on commodity hardware, within four seconds. We also analyzed 42 malware samples from seven different malware families, that are known to be "silent" in a virtualized or emulated environments, and all of them showed their true malicious behavior within our bare-metal analysis environment.

Keywords: bare metal, dynamic malware analysis, system restore, VM-aware

1. INTRODUCTION

The use of virtualized environments is ubiquitous in malware analysis. The ability to isolate and quickly restore the system to a known configuration after an analysis run are two key features of virtualized environments that facilitate malware analysis. Furthermore, during the first quarter of 2011 alone, McAfee Labs have identified more than six million unique malware samples [1]. This ever-increasing flood of new malware demands high performance analysis frameworks that can inspect as many malware samples as possible within a given period of time. Virtualization-based solutions have been an obvious choice. Unfortunately, malware authors have developed a new class of malware, called VM-aware malware, which can detect virtualized or emulated environments and subsequently change its behavior to foil the analysis.

The environment detection techniques used by virtual machine (VM) and emulation-aware malware are well-known and well-described in the literature [2,3]. These techniques exploit some of the software or hardware artifacts introduced by the virtualization or emulation layer between the running operating system and the hardware. For example, some of the detection techniques identify in-guest, system-specific configurations or specific I/O ports, while others rely on inaccurate system emulation or sophisticated time-based detection attacks [2,4]. For instance, a system running inside a VMware guest OS can simply inspect the names of the available virtual devices, read the magic I/O port (`0x5658, 'VX'`), or simply read the value of LDTR or IDTR registers (using unprivileged SIDT or SLDT instructions), which will be different from the known values found on a bare-metal system [5]. All of these detection techniques can be executed in *usermode.*

More transparent, hardware-assisted, virtualization-based malware analyzers, such as Ether [6] and Azure [7] have been proposed. Although these systems reside outside of the guest OS, they share the underlying hardware CPU with the virtualized guest OSes. As a result, all privileged instructions must be properly intercepted and virtualized by the virtual machine monitor (VMMs). This consumes more CPU cycles per each instruction, allowing the detection of the analysis system. For example, certain detection techniques can measure the difference of execution times of privileged instructions between real and virtualized machines [3,4] or the ratio of the execution times between privileged and unprivileged instructions in a single virtualized machine [2]. It was proposed that timing attacks can be thwarted by intercepting timing instructions and providing disguised timing information. However, Garfinkel et al. [8] have argued that this is not always possible, and indeed nEther [4] has demonstrated in-guest detection of the Ether platform using local timing attack.

Another approach to transparent malware analysis is based on the whole-system emulators, such as QEMU [9]. In theory, one should be able to correct all CPU emulation bugs in the emulators to avoid detection based on inaccurate system emulation. For instance, Anubis [10], which is based on an instrumented QEMU, randomizes hardware IDs and keeps patching known CPU emulation bugs. However, it is always possible that new bugs will be found, resulting in an arms race. In fact, Paleari et al. [11] have demonstrated an automated technique to produce thousands of such attacks to detect CPU emulation inconsistencies.

There has been a considerable amount of research focused on the detection of environment-sensitive malware by identifying differences in their execution behavior within a virtualized system and a *reference system* [12–14]. The reference system is ideally an OS running on real hardware, inside which VM-aware malware samples show their true malicious behavior. However, all previous systems have used either virtualized or emulated systems as the reference systems to evaluate their techniques. A sophisticated malware can detect all such incomplete realizations of the reference systems and hide its malicious behavior, showing no difference in their execution behavior [12].

The definitive way to observe the actual behavior of any unknown malware is to execute the sample in a real-hardware-based system, also called a bare-metal system. This approach entirely eliminates the possibility of modified behavior based on the detection of a virtual environment. However, this approach poses several practical challenges. One of the critical challenges is the ability to *efficiently* restore the analysis environment into the initial *clean state* after each malware infection. Although there exist both hardware and software-based solutions that restore the hard disk state, these solutions require a system reboot [15, 16]. System reboots are inherently slow as they are heavily I/O bound. Starting from the initial execution of the Power-On Self-Test (POST), several devices need to be initialized by the BIOS, and the corresponding drivers initialized by the operating system. Also, other OS components, including the malware analysis parts, have to be reinitialized after every reboot. In our experience, a typical system on a commodity hardware takes more than 15 seconds to reboot. We believe that the downtime required for a reboot is a major impediment towards the deployment of automated malware analysis frameworks on bare-metal.

In this paper, we propose a bare-metal analysis framework based on a novel technique that allows reboot-less system restoring. This framework, called BareBox, executes and monitors malware run in a native environment. After an analysis run has been completed, the running OS is restored back to the previously-saved clean state on-the-fly, *within a few seconds*. This restoration process recovers not only the underlying disk state but also the volatile state of the entire operating system (running processes, memory cache, filesystem cache, etc). Our restoration system consists of a overlay-based volatile mirror disk, a symmetric physical memory partition, and a small custom-written operating system, which we call *Meta-OS*. *Meta-OS* restores the entire physical memory of the running operating system once a malware analysis run completes. Because of the circular dependencies between the physical memory and the CPU context, it is practically impossible to perform a complete OS restore from within the OS that is being restored. To resolve this problem, *Meta-OS* provides *out-of-OS* execution control. The operation of other devices and peripherals also closely depend upon the state of the main physical memory. Careful handling of the device states is required to safely restore the state of the OS.

As the majority of malware targets Windows systems, we have chosen an x86-based Windows XP system as the target for implementing our prototype. However, the concept can be easily extended to other operating systems.
This work makes following main contributions:

- BareBox can efficiently profile the true behavior of malware by actually executing it in a native hardware environment.
- BareBox can restore a running operating system on commodity hardware back to its previously-saved state faster than any existing solution (within four seconds on average). The reason is that no reboot is required.
- Restoring the volatile state of the operating system (running programs, memory cache, filesystem cache, etc) makes it possible to quickly reproduce identical system states for repeated security experiments in a bare-metal environment. This is trivial for a virtualized system, but required the development of a small, custom OS to perform this on bare metal.

2. GOALS AND CHALLENGES

In this section, we describe the main goals and challenges of the proposed system.

2.1 Goals

No virtualization.

Our primary goal is to develop a bare-metal framework for analyzing malware that does not rely on virtualization or emulation techniques, such that all of the VM detection techniques used by malware are rendered ineffective. This excludes the possibility of utilizing hardware assisted virtualization such as the Intel VT and AMD-V technologies.

Efficient environment restore.

The environment has to be restored after each malware analysis run. Speed and efficiency of the restoration technique largely affects the effectiveness and throughput of the entire analysis framework. We want to restore the environment as fast as possible to the point where the next analysis can be initiated.

Malware profiling.

The system should be able to monitor a malware execution in the bare-metal environment. In particular, monitoring should include the interactions between the malware program and the operating system.

2.2 Challenges

In this section we describe some of the major challenges when trying to analyze malware on a bare-metal system.

Isolation.

Although the whole purpose of the bare-metal analysis system is to create an environment that closely resembles a "real" system, there are unavoidable risks when executing unknown malicious code on real hardware. With direct access, there is the possibility for serious system-level corruption, for instance, by malicious updates to the BIOS. Complete isolation of BIOS and peripheral devices, such as the hard disk, is not as trivial as it is in a virtualized system.

Stealthiness.

Bare-metal based analysis systems are definitely stealthier than any VM/emulation-based analysis systems with respect to VM-aware malware. However, the analysis component must reside inside or next to the operating system running on bare metal to extract analysis information. Theoretically, complete transparent analysis is impossible if the malware runs at the same privilege level as the analyzer [6]. One practical approach is to be undetectable from user-mode malware by implementing stealth rootkit techniques or by employing security by obscurity (for instance, randomizing the signature and/or the location of the analysis components). Another approach can be a *black box* approach where a malware sample is executed on bare metal without the presence of any in-guest analysis component. In this case, the analysis focuses on inspecting network traffic from outside and later checks for changes in the system state and configuration. Although this approach is stealthy, a lot of the contextual and dynamic information about the behavior of a malware sample is lost.

System Restore.

Without virtualization, options available for fast and efficient system restore solutions are limited. Most of the available solutions only restore the operating system's persistent state by restoring the hard disk, which then requires a system reboot to complete the restoration process. System reboot is heavily I/O bound, which causes long downtimes between subsequent analysis runs. Restoration of BIOS and other device firmwares are yet another challenge.

3. SYSTEM OVERVIEW

3.1 Threat Model

Without any in-guest presence, it is very difficult to perform detailed dynamic malware analysis in a non-virtualized, bare-metal environment (unless some specialized external hardware instrumentation is devised to directly monitor the memory and CPU). As our goal is to analyze malware in a bare-metal system on commodity hardware, we consider this inevitable trade-off of in-guest-presence as acceptable.

Malware running at ring0 can always detect the presence of BareBox analysis component using a memory scan. By forcefully mapping the reserved extended area of the physical memory, such malware can even corrupt the Meta-Os or the saved snapshot. However, mapping of reserved physical memory area is unusual and can be considered suspicious. To this end, the BareBox framework focuses only on the analysis of user-mode malware by disabling new kernel module loading, and preventing usermode access to kernel memory. The framework hides its presence from the user-mode malware using rootkit-like techniques. As effective restrictions for loading arbitrary kernel modules became operative in recent operating systems, such as Windows 7, an overwhelming portion of the present-day malware is user mode malware.

Attacks using return-oriented techniques [17] or kernel-exploits can potentially bypass the restrictions. In such cases, the most effective recovery would be an off-line disk-restore based recovery. As of now, an internal timeout for each analysis run is enforced, whose expiration forces a trap into the *Meta-OS* to perform a physical memory clean up.

3.2 General approach

Our approach to quick system restore is based on the symmetric partition of mutable resources such as disk and memory. In this scheme, one of the partitions is used to preserve a *snapshot* of the system's state. A *snapshot-save* operation creates a restorable state of the running operating system. This state can be restored back to a previous snapshot by a *snapshot-restore* operation. The target OS is installed on the main disk, and a disk identical to the main disk is attached to operate as mirror. Access to the disk is controlled by a software-based overlay mechanism that selectively redirects sector read and write operations to either the main disk or the mirror disk.

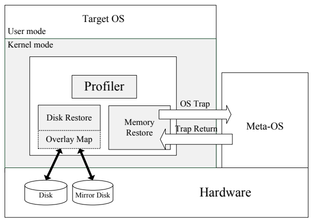


Figure 1: Architecture Overview

The physical memory is partitioned in such a way that one of the partitions can be used as a snapshot area. The other partition stores the target OS, where the actual malware is executed. During a *snapshot-save* operation, the entire physical memory corresponding to the target OS is copied over to the snapshot area. Live state of the running operating system is restored by restoring the physical memory of the operating system from the previously-saved snapshot. With this approach, only one snapshot point can be saved at a time, but the system can be restored back to this snapshot as many times as needed.

Complete restore of the running operating system involves the overwriting of critical OS structures residing in physical memory (e.g., the page table, interrupt handlers, etc.) and the restoring of the CPU context, including system-critical registers (e.g., IDTR, GDTR and other control registers). These registers are highly coupled with the content of the physical memory. We explain in detail in Section 4 why this restoration process can only be done from an *out-of-OS* execution. We implemented a small operating system, called *Meta-OS*, that resides outside the physical memory of the target operating system. *Meta-OS* creates the physical memory snapshot of the target OS and later restores it.

4. IMPLEMENTATION DETAILS

4.1 Disk Restore

A disk restore simply requires canceling all changes made to the disk by write operations. When we cancel only those changes that are made after a certain point in time, called the snapshot point, we effectively restore the state of the disk to that particular snapshot point. We achieve this by proper redirections of read and write operations to the main and the mirror disks. A *snapshot-save* operation simply implies that all further write operations to the main disk are redirected to the mirror disk. With this redirection in place, we effectively freeze the contents of the main disk. However, all read operations are still forwarded to the main disk, except those read operations to particular sectors that were previously redirected and written to the mirror disk. Such reads to "dirty sectors" are served from the mirror disk to reflect the changes made to the main disk after a *snapshot-save* operation.
We implemented two methods for the storage of the mirror disk; a RAM disk and a physical hard disk.

RAM disk.

In this technique, the mirror disk is implemented as a RAM disk so that no external hardware is required. The maximum amount of changes to the main disk is limited by the size of the RAM disk, which in turn, depends on the size of the available system memory. Although not suitable for lengthy analysis sessions that generate large amounts of disk write operations, this technique is very effective for analyzing malware that is executed for short periods of time.

Hard disk.

For this implementation, two identical disks are required to be installed in the machine, similar to a RAID1 configuration. One disk contains the OS, while the other disk (the mirror disk) is used as a temporary storage. As the disks are identical, they can have a sector-level one-to-one mapping, allowing us to redirect any number of write operations from the main disk to the mirror disk.

An overlay map for the book-keeping of the disk write operations is required to selectively redirect read operations for dirty sectors on the main disk to the mirror disk. This overlay map needs to be in the memory for efficient disk I/O. However, the size of the map gets quite large even if we use a single bit per sector mapping (e.g., a 256 GB drive with 512-byte sector size, requires 64 MB overlay map). To reduce the size of the map, we aggregate consecutive sectors into chunks. The granularity of the dirty-sector-write is a chunk, instead of individual sectors. We found that the average number of sectors accessed at once during a normal usage is 80 sectors or 40 KB. We chose a chunk size of 32KB for efficient bit-shift calculations of chunk locations. For a 256 GB drive with a 32KB chunk size, our mechanism only requires 1MB for the overlay map.

To keep the consistency of the underlying filesystem, the OS is forced to flush its file-system-cache before taking a snapshot. We perform this by sending appropriate device control codes to all logical volume devices. Disk operation redirection is implemented by intercepting the low-level disk driver major functions. Every access to the disk from higher-level drivers, such as the file system driver, volume and partition manager, page-fault memory manager, etc., are directed through this disk driver.

4.2 Memory Restore

The disk restore component can restore the disk image on-the-fly by simply resetting the overlay map. Unfortunately, this is not enough. The modified state of the disk is still cached within several data structures of the operating system in memory, file system structures being the most prominent. Also, there can be several open file handles or memory-mapped files related to new files created after a snapshot point. Therefore, an on-the-fly restore of the underlying disk will induce serious inconsistencies between the in-memory filesystem structure and the actual in-disk structures, resulting in data corruption. This means that one cannot simply restore the disk on-the-fly. Instead, a system reboot is required for a clean restore. As a result, volatile state of the running processes is lost after the reboot. However, our approach to physical memory restoration makes the rebootless restore of a bare-metal system possible.

4.2.1 Physical memory partitions

The available physical memory is partitioned into three parts. The operating system is loaded into the first part, which starts from the absolute hardware address zero. The second part of the physical memory is used to take a snapshot of the first part. Finally, the small operating system *Meta-OS* resides in the third part of the physical memory. We implemented this component as a kernel module that is loaded into the target OS.

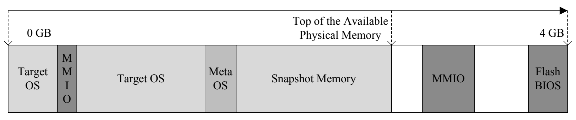


Figure 2: Physical memory allocation

In the x86 architecture, the usable physical memory address space is not entirely contiguous. Two types of devices are mapped into the processor address space - actual physical memory devices (RAM) and other memory-mapped I/O (MMIO) devices such as AGP and PCI cards. For 32-bit compatibility, the BIOS has to map such devices within the 4GB addressable range of the processor address space. In a machine with 4GB of RAM memory, the MMIO device mapping creates an unusable region of the physical memory usually known as the "PCI hole". *Meta-OS* needs to resolve such resource conflicts by carefully relocating and resizing the memory partition. This leads to effectively four types of memory sections: The target OS memory, the *Meta-OS* memory, the snapshot memory, and the hardware-reserved memory section (shown in Figure 2).

4.2.2 Need of out-of-OS execution control

When the operating system is fully initialized, the CPU is in protected mode with paging enabled. Memory restoration needs to be performed when the CPU is in this state. Before we explain the details about the memory restore mechanism, we briefly review the basics of the protected mode CPU execution.

When the processor is in protected mode with paging enabled, every single memory reference made by the executing instruction is treated as a virtual address, which must be translated into a physical address using the Global Descriptor Table (GDT) and the page table. These tables reside in memory, and the processor knows their locations from two registers, the GDTR and CR3, respectively. For example, if the memory reference is made for the code segment to execute an instruction, the CPU needs to first find the offset of that segment. Unlike in real mode, during protected mode execution, finding the segment offset is a two-step process. The CS segment register only provides an index into the GDT table, and the actual segment offset is retrieved from the corresponding entry in the GDT. Once the effective virtual address is computed by adding the segment offset, it is translated into the physical address using the page table. The appropriate page table is retrieved from the page directory, whose physical address location is stored in the CR3 register. Note that the location of the GDT stored in the GDTR resister is actually a virtual address, which must first be translated into a physical address to access contents of the GDT table for the initial segment offset calculation.

To copy and restore the physical memory, it must be first mapped into virtual memory using page tables, as this is the only way of accessing memory while the CPU is in protected mode. In the x86 architecture, restoring the physical memory becomes challenging because it involves overwriting both the GDT table and the page table, which, in turn, are used to translate virtual addresses into physical memory locations. This includes the current location (virtual address) of the code (EIP) that performs the memory restoring. That is, the memory restore code would change memory mappings that interfere with its own execution. Because of these circular dependencies, it is impossible to restore physical memory of a live operating system from within the same operating system (with arbitrary physical memory content). For this reason, we have implemented a small operating system, called *Meta-OS*, as a memory restore component that resides outside of the physical memory of the target OS.

4.2.3 Meta-OS

Meta-OS is loaded into the extended area of the physical memory. This area of the memory is marked as unavailable (reserved) to the target OS during the boot time. Thus, the *Meta-OS* loader has to directly modify the page tables to forcefully map this area of the physical memory. A specific software interrupt or a device I/O control request triggers the *snapshot-save* and *snapshot-restore* operations. These operations involve operating system context switches from the target OS to the Meta-OS and back to the target OS. Interrupts are disabled during this process, and non-maskable-interrupts (NMI) are simply passed while the CPU is executing in the context of the *Meta-OS*. During any other time, *Meta-OS* remains dormant.

4.2.4 OS context switch

An OS context switch involves switching the Segment Descriptor Table (SDT), the Interrupt Descriptor Table (IDT), the page table, and other CPU register contexts. The final execution control transfer is implemented using the x86 interrupt handling mechanism. As both operating systems are running in the same privilege level (ring0) with separate Interrupt Dispatch Tables (IDTs), once the execution control is transfer from one OS to the another, the previous OS will have no way to force the control transfer back to itself. Thus, all OS context switches are voluntary.

SDT and IDT pointers are stored in the GDTR and IDTR registers, which can be modified simply using LGDT (Load GDT) and LIDT (Load IDT) instructions, respectively. A page table switch is achieved by modifying the CR3 control register, which holds the physical address of the page directory [18]. The page table switch operation forces the CPU to flush the TLB and the instruction pipeline cache. At this point, the new page table must contain a correct mapping for the virtual addresses of at least the GDT and the instruction pointer (EIP). A similar situation occurs when switching the CPU from physical to virtual address mode and during process context switches. Physical to virtual address mode-switch is typically handled by a one-to-one mapping between physical to virtual addresses. In case of the process context switches, OS maintains an identical kernel page map for each process. We follow the strategy of process context switch and copy the current page table to the *Meta-OS* page table before switching to it.

Switching back from *Meta-OS* is trivial after a *snapshot-save* operation because both page tables are identical. However, this is not the case while switching back after a *snapshot-restore* operation. Since page tables are not identical, simply switching back to previous table will cause the CPU to reference an invalid GDT structure, resulting in a hard reset of the CPU. Meta-OS handles this situation using a combination of the following techniques.

GDT relocation.

Before switching to the target OS page table from the *Meta-OS* page table (after a *snapshot-restore* operation), we have to make sure that the page table transition will be successful. Since the page table of the target OS is the restored version of the previous snapshot, we are not allowed to modify them in a way such that current GDTR will eventually point to a valid structure. Instead, one has to relocate the current GDT of *Meta-OS* to the previously-saved GDTR address of the target OS. We do this by first mapping the physical address of the *Meta-OS* GDT to the previously-saved target OS GDTR address (a virtual address), and then switching the GDTR to that address. By doing this, we guarantee that even after switching to a previously-saved page table with arbitrary mappings, the GDTR will still point to the proper descriptor table.

Segmented Address Space.

Although, GDT relocation technique maintains the proper descriptor table after the page table switch, the instruction pointer will still hold the address of the *Meta-OS* address space. In this case, the next instruction execution depends on where the restored page table of the target OS map this address to. Since there is no atomic instruction to simultaneously update both CR3 and EIP registers, this transition has to be handled incrementally. Note that the restored descriptor table and the page tables are in fact the same tables that were copied to *Meta-OS* page tables during the previous *snapshot-save* operation. However, the *Meta-OS* code segment might be mapped to different virtual address at each OS context switch. *Meta-OS* uses segmented memory address model for its code segment, such that the instruction pointer becomes an offset address, rather than the actual virtual address. By this approach, even after switching to an old page table having different base address for *Meta-OS*, the instruction pointer (EIP) address is still translated into correct *Meta-OS* instructions. These instructions complete the proper execution control transfer to the restored target OS.

Since we disable interrupts during snapshot operations, we cannot use the interrupt-based DMA controller for copying physical memory. Instead, we achieve fast memory copy by enabling the cache attributes in the hardware page tables that are used to map the physical memory into a linear virtual address. Even if the TLB miss rate is high as *Meta-OS* swipes across the address space, this type of cache-enabled page mapping helps to fully utilize the L1 and L2 CPU caches during memory copy operations. In our case, this technique improved the performance by more than ten times. We achieve additional performance improvement by deploying the pre-unrolled execution of the memory copy instructions [19]. The main idea of this technique is trying to fill the CPU instruction pipeline with multiple memory move instructions.

4.3 Device Restore

Beside disk and memory restore, a complete restore of a live running system requires that all its devices are restored to the previous snapshot state as well. This is also necessary because the state of some devices depends on the contents of the physical memory. This requirement is difficult to satisfy, as the exact definition of "device state" is device-specific. To deal with this problem, we manipulated the device power state, which in turn forces a reset to the internal device state. A device is set to *Sleep Power State* (or *Off Power State* if the device does not support the sleep state) before *snapshot-save* operation is started. Its power state is restored back after the snapshot process completes. Similar power state manipulations are performed before and after the *snapshot-restore* operation. This way, instead of saving and restoring the internal states of the devices, which is in fact challenging, we forcefully reset/restart those devices.

Manipulation of the device power state is a time-consuming process, while one of our goals is to achieve the fastest possible system restore. By assuming that certain devices do not change their state during the analysis, we could exclude such devices from power state manipulation and considerably speed up the restore process. However, this assumption does not hold for all PCI devices, such as network adapters. In our selective device restore experiments, we could safely exclude several PCI devices, such as the hard disk controller, PCI bridge, memory controller, SMBus controller, graphics controller etc. Although we skipped the power state manipulation of the graphics controller, each *snapshot-restore* operation forces a video memory refresh operation, if the video-memory is mapped to the physical memory. However, to this end, we do not restore the internal memory of the GPU.

4.4 Execution Monitoring

We also implemented a prototype malware monitoring component. This component hooks the System Service Descriptor Table (SSDT) to record the interactions of malware with the operating system by tracing system calls. System call arguments of important calls are automatically extracted along with their semantic information.

We disabled new driver loading during the analysis to protect the integrity of the monitoring component, even though driver load could be reverted by the physical memory restore. As malware can potentially scan the physical memory using `\device\PhysicalMemory` or using the undocumented API `NtSystemDebugControl` to detect kernel level hooks, we disabled both scanning techniques by instrumenting related system calls.

The monitoring component of BareBox receives new malware samples from an external controller through the network. The recorded analysis data is streamed back to the controller, where it is recorded in a database.

5. EVALUATION

In this section, we present the evaluation of our system. All experiments were conducted on a freshly-installed Windows XP (Service Pack 3), running on bare-metal commodity hardware (3.60GHz Intel Pentium 4 processor with 1GB Memory). Two 256GB hard drives were installed as a main and a mirror disk.

5.1 Restore Verification

We first verify that the disk operation redirections consistently reflect the possible changes made to the system. This is important, as we need to allow the malware to execute properly during analysis. In the second part of the experiment, we verified that all such changes are successfully restored by our system.

5.1.1 Disk restore verification

To verify disk restoration, we calculated the MD5 hash of all files in the *Windows\System32* directory in the clean state of the system. A snapshot of the system was taken using the *snapshot-save* operation. After that, we made a copy of the *Windows\System32* directory and again calculated the MD5 hash of all files. We compared the file hashes and found them to be identical. Therefore, we verified that the redirected read/write operations work properly. To verify that new/changed files can be properly mapped as memory mapped files, we executed some of the applications from the copied version of the *Windows\System32* directory. The applications were executed without any issue. Note that, while executing an *.exe* file, the Windows subsystem first maps the executable as a memory mapped file before executing it.

In the next step, we resized the existing partition of the disk and created a new partition to effectively modify the Master Boot Record (MBR) of the disk. We also deleted all existing system files from the system root C:\directory (except locked paging and hibernation files). This action will cause the system to fail to boot after a restart.

We executed the *snapshot-restore* operation, and the disk was restored to the previous clean state with a single partition that contains all the deleted system files intact but without the copy of the *Windows\System32* directory. To ensure that the restore is permanent, we performed a cold reboot of the system. The disk was verified to be in the previous, clean state.

5.1.2 Volatile state restore verification

In this experiment, we wanted to verify that the volatile state of the running system, including running processes, open file handles, etc., are properly restored.

Before executing a *snapshot-save* operation, we started multiple applications from the *Windows\System32* directory and then saved the list of running process and loaded drivers. After taking the snapshot we installed a DKOMbased FU.rootkit malware driver. We logged off from the current user session, effectively terminating all the running applications, and logged in as a different user. We initiated the *snapshot-restore* operation from that user session, and within seconds, the system was restored back to the previous user's active session. All the running processes were intact but without the rootkit driver loaded in the system.

5.2 Performance

We compared the performance of our system with different system restore options available for bare-metal and virtual machines systems. We did not test a hardware-based disk restore option (such as Juzt-reboot [15]), but since all of them require a system reboot, we simply measure the system reboot time of the bare-metal system and use this as a lower bound. To measure a fast reboot with commodity hardware, we installed a Windows XP system on an Intel quad-core CPU with 4GB RAM and a 48GB solid-state drive (SSD) with average throughput of 450MB/s. We ignored the initial BIOS initialization of the machine, as it is largely depended on its hardware configuration (e.g., for the above mentioned configuration, the BIOS initialization time was more than 7 seconds because of the custom configuration prompt of the SSD drive). We only measured the time of the soft-reboot of the operating system. That is, the time duration from the operating system boot menu to a completely logged on session. This will typically underestimate the actual time that a system requires to perform a soft-boot.

By profiling the time required for different stages of BareBox' restore process, we found that a significant amount of time was required for device power state manipulation. Thus, we performed experiments by manipulating the power state of only a subset of attached hardware devices. In particular, we performed three sets of experiments:

- All Devices: All devices that are suspended by the system during a suspend-to-RAM operation are included.
- Minimum Devices: Only those devices that are required for the malware analysis framework were included.
- Memory Only: No device related operations were performed. The purpose of this experiment was to measure the memory copy performance because not all devices supported this type of memory restore.

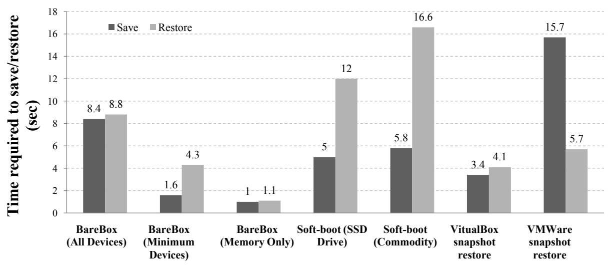


Figure 3: Evaluation of different system restore techniques.

Figure 3 shows the times that it takes to save and restore snapshots for different configurations. This includes BareBox, performing a soft-boot on bare metal, and virtual-machine-based system restore solution. Looking at the results, we can see that the performance of BareBox is comparable to fast, virtual-machine-based solutions. For example, in the required "minimum device" configuration, BareBox is as fast as VirtualBox. The difference, of course, is that BareBox restores the system on bare metal. Alternative mechanisms to restore the system on bare metal take significantly longer (at least three to four times as long).

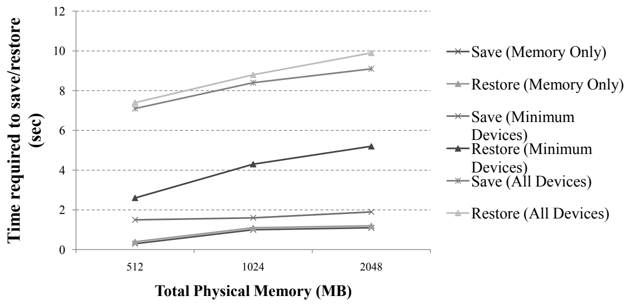


Figure 4: BareBox restore compared with different device restore options.

We also measured the system restore performance of BareBox with different physical memory sizes. As shown in the Figure 4, with a minimum device configuration, the time required for a restore is about four seconds on average. This is about three times faster than booting the system from an SSD-based hard disk.

5.3 Dynamic Malware Analysis

A malware analysis system can be evaluated based on three main factors - quality, efficiency, and stealthiness. Quality refers to the richness and completeness of the analysis results produced. Efficiency measures the number of samples that can be processed per unit time. Stealthiness refers to the undetectability of the presence of the analysis environment.

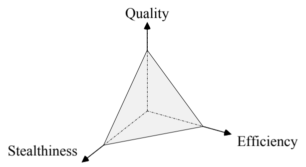


Figure 5: Malware analysis framework evaluation

There is a constant tension between these factors, which is represented as a triangular surface in Figure 5. That is, while trying to achieve better results for one factor, the analysis technique has to make compromises in remain two other factors. For example, an analysis system with an *in-guest* agent tends to produce high quality analysis results but it is less stealthy and prone to subversion. VMM and emulator-based *out-of-OS* analysis systems are stealthier against in-guest agent detection, but because of the *semantic gap*, some level of introspection is required, using the domain knowledge of the target OS [10, 20]. This makes analysis less efficient and limits the quality of the produced results. Quality and efficiency trade-offs between the fine-grained and the coarse-grained analyses are well described in the literature. BareBox aims to achieve high efficiency while producing results of good quality, which, however, is against the constraints discussed above. A particular focus is on being stealthy when facing VM-aware malware.

5.3.1 Stealthiness

Dinaburg et al. [6] have proposed five theoretical requirements, for building dynamic, transparent malware analysis system, with absolute stealthiness. By executing malware on bare-metal, BareBox immediately satisfies three out of five of these requirements, namely *Identical Basic Instruction Execution Semantics*, *Transparent Exception Handling*, and *Identical Measurement of Time*. We maintain the requirement of *Higher Privilege* by disabling the loading of new kernel modules. As an in-guest analysis system, it is hard to satisfy the requirement of *No non-privileged side effects*. We hide the presence of our analysis component from user mode applications by instrumenting file and memory-related system calls.

For the evaluation of the BareBox with VM/emulation-aware malware, we collected 200 malware samples from the Anubis [10] database that are known to detect virtualization and emulation environments. To maintain the diversity of the sample set, we included six samples per malware family in our actual experiments. Seven different malware families were included, as identified by several major anti-virus vendors such as Kaspersky, McAfee, and ESET. For each family, we tried to include different versions of the malware, if available. We executed them inside a virtualized environment (VMware), an emulated environment (QEMU), and in our bare-metal system and compared their system call traces. The same system call monitoring component was used for all analysis environments.

Malware execution is bound to be slower within virtualized or emulated environment, which might result in smaller number of observed system calls per unit time. However, we assume that one minute, the execution time allowed per malware sample, is enough for the above-selected malware families to show malicious behavior through network interaction and/or new-process creation, regardless of the slowed execution. In fact, we executed some benign programs on these analysis environments, and we observed almost exactly the same number of network and process-creation related system calls within the one minute of execution.

Table 1: Interactions with the Network

Malware Family	BareBox	VMware	QEMU
Rebhip	346	10	55
Telock	205	107	34
Conficker	24	20	16
Zlob/Zeus	408	406	176
Sdbot	152	45	30
Agobot	50	48	3
Armadillo-4	172	82	58

We first wanted to see how much more network activity can be observed in BareBox compared to virtual environments. There are no network-specific system calls for Windows XP. Instead, the network communication is handled by IOCTL calls to the network-endpoint-related file objects. Thus, we monitored calls to these endpoints to assess the network activity of our malware samples. The results in Table 1 clearly show the increased number of network related system calls in our BareBox environment. In addition, we checked how many new processes were created by the malware samples, depending on their environment. As can be seen in Table 2, BareBox was able to elicit more process-creation activities across the board.

Table 2: Number of new process creation

Malware Family	BareBox	VMware	QEMU
Rebhip	9	0	3
Telock	2	1	1
Conficker	0	0	0
Zlob/Zeus	10	10	4
Sdbot	4	1	1
Agobot	50	3	1
Armadillo-4	1	0	0

5.3.2 Efficiency

For a given hardware configuration, usually a program is executed most efficiently when run on bare metal, which is true for the execution of malware programs as well. In addition, while using available disk-restoration-based solutions, the system has to not only reboot but it also requires to load the analysis framework after each reboot. Our system does not have this limitation as the restore operation also restores the state of the analysis component that is ready to analyze the next malware.

To compare the overall efficiency, we measured the overall throughput of each analysis system. We allowed each sample to load and execute for an arbitrary time, 15 seconds, and then the system was automatically restored back for the next run. Virtualized and emulated environments were restored using corresponding snapshot restore mechanisms. To compare with the disk-restore based system, we used the disk mirror component of the BareBox for preserving the disk-state. The system was rebooted after each analysis run to restored the system. We performed an automated analysis of 100 samples in each system to assess the average throughput.

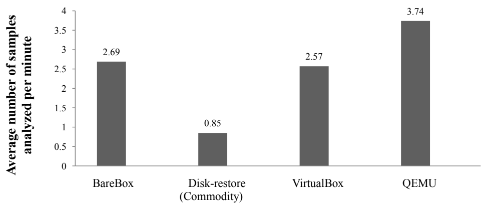


Figure 6: Malware analysis throughput

Figure 6 presents the average number of malware samples analyzed per minute. Compared to the disk-restore based system on commodity hardware, BareBox improved the throughput threefold. Slightly better performance of the BareBox compared to the VirtualBox-based system can be attributed to the expedited initialization of the network device. We observed a slightly longer delay in network device initialization in case of VirtualBox, after a snapshot restoration. Fully working network interface is essential to initiate a next run of the malware analysis.

5.3.3 Quality

BareBox analysis is limited to system calls only. However, without executing direct or indirect system calls, it is almost impossible to comprehend malicious activities. Since malware is executed on bare-metal, we expect the behavior observed by BareBox to be more close to the actual behavior of the malware exhibited in the wild. Being an in-guest analyzer, our monitoring component can access operating system internals and potentially extract all semantic information of the malware execution. For example, it can initiate additional system calls in the context of executing malware to extract more contextual information. BareBox does not provide fine-grained analysis. Although, it is possible to implement capabilities like user-mode API monitoring and debugger-like fine-grained analysis to improve the quality of the analysis, it compromises BareBox's efficiency and stealthiness.

6. LIMITATIONS

The efficiency of BareBox's rebootless restore largely depends on the actual hardware configuration of the system. For instance, a system with many hardware devices will take longer to restore because BareBox has to force a change in the power state of all of these devices. Currently, BareBox does not restore the entire internal state of attached devices (such as base address registers (BARs)). Instead, it assumes that they are not changed during the short analysis period. This assumption is reasonable, as such states are rarely changed once adjusted by theOS during the initial boot process. For obvious reasons, the restore system cannot restore the changes made to the external resources such as network communication.

Chen et al. have developed a detailed taxonomy of evasion techniques used by malware against dynamic analysis systems [21]. Based on their taxonomy, although BareBox is not detectable through *Hardware*, *Application*, and *Behavior* (timing) based evasion attacks, not all *Environment*-based attacks are defeated. For example, attackers could perform network resource fingerprinting using an external agent (connecting to remote server to identify the host network). Moreover, to prevent the detection of in-guest memory presence and to maintain privilege isolation from the executing malware, BareBox instruments related system calls and disables loading of new kernel modules. Such restriction itself, although not an unconditional indication of BareBox, could be used as a non-privileged side effect for detection. When automating dynamic malware analysis, even a fully transparent analysis system typically requires some form of in-guest agent to receive and initiate execution of a new malware sample. This problem can be mitigated by completely relying on OS-provided RPC mechanisms or implementing a self-destructing agent that removes itself from the system after launching the malware payload. BareBox currently does not implement either of the mechanisms.

Our system must allocate a finite analysis time for each inspection. Malware can potentially exploit this limitation using a *delayed infection* technique. For example, during the initial execution, a malware can choose to stay dormant for long enough so that it passes the allocated, finite analysis time without revealing any malicious activities. Time-triggered or condition-triggered infections (logic bombs) are likely to bypass our dynamic analysis.

7. RELATED WORK

System restore.

AVMM [22] has implemented a physical memory partition technique to improve the efficiency of virtualized clients. Some of the related work on fast reboot and restore are primarily based on preserving system cache. Otherworld [23] uses microreboot techniques to quickly recover applications in case of a kernel crash. This is done by initiating another kernel, called crash kernel, and restoring the state. *Warm-cache-reboot* [24] leverages virtual machine monitor (VMM) technology to preserve the page cache and to help restore the operating system after a reboot. RootHammer [25] reuses the previous VM image from memory, and Recovery Box [26] uses non-volatile memory for fast system restore. However, these systems are focused on fault tolerance and are not designed for restoring the OS to some previous snapshot after a number of modifications to the system's volatile and persistent states, such as hard disk contents. Also, most of these quick restore solutions rely on VMM technology, which is not compatible with our threat model.

Transparent analysis.

For more transparent analysis of malware, researchers have implemented many *out-of-OS* analysis systems that eliminate the *in-guest* presence of components. Some systems are based on whole-system emulation (e.g., Anubis [10], Norman Sandbox [27]), while other systems leverage hardware-assisted virtualization technology to achieve transparency (e.g., Ether [6], VMwatcher [20]). Although there is no *in-guest* presence with these analysis system, nEther was able detect Ether [4], while Paleari et al. were able to automatically produce hundreds of detection codes for emulation based systems [11].

Evasion detection.

Researchers have explored different techniques to detect evasive malware behaviors. Lau and Svajcer have employed a dynamic-static tracing system based on an instrumented Bochs virtual machine to identify VM detection techniques used inside packers [14]. Kang et al. [13] uses a trace matching algorithm to locate the point of execution diversion between an emulated and real environment, and dynamically patching the program to make it behave as observed in a reference system. Balzarotti et al. [12] proposed a system for detection of split personality malware based on the deterministic replay of system call traces generated in a reference system. All of these evasion detection techniques require an ideal *reference system*. BareBox can serve as such a reference system and, hence, complements previous work.

8. CONCLUSION

In this paper, we presented BareBox, a framework for dynamic malware analysis in a bare-metal environment. To facilitate efficient analysis, we introduce a novel technique for reboot-less system restore. Since the system executes malware on real hardware, it is not vulnerable to any type of VM/emulation-based detection attacks. We evaluated the effectiveness of the system by successfully monitoring the true malicious behavior of VM/emulation-aware malware samples that did not show malicious behavior in emulated or virtualized environments. After each such analysis, our system was able to efficiently restore the bare-metal system so that the next analysis could be initiated.

Acknowledgments

This work was supported by the ONR under grant N000140911042, the ARO under grant W911NF-09-1-0553, and the National Science Foundation (NSF) under grants CNS-0845559 and CNS-0905537.

9. REFERENCES

[1] M. Labs, "Mcafee threats report: First quarter 2011," McAfee, Tech. Rep., 2011. [Online]. Available: https://secure.mcafee.com/us/resources/reports/rp-quarterly-threat-q1-2011.pdf

[2] T. Raffetseder, C. Kruegel, and E. Kirda, "Detecting system emulators."

[3] P. Ferrie, "Attacks on virtual machine emulators," Symantec Corporation, Tech. Rep., 2007.

[4] G. Pék, B. Bencsáth, and L. Buttyán, "nether: in-guest detection of out-of-the-guest malware analyzers," in *Proceedings of the Fourth European Workshop on System Security*, ser. EUROSEC '11. New York, NY, USA: ACM, 2011, pp. 3:1–3:6.

[5] J. Rutkowska, "Red pill... or how to detect vmm using (almost) one cpu instruction," 2004. [Online]. Available: http://invisiblethings.org/papers/redpill.html

[6] A. Dinaburg, P. Royal, M. Sharif, and W. Lee, "Ether: malware analysis via hardware virtualization extensions," in *Proceedings of the 15th ACM conference on Computer and communications security*, ser. CCS '08. New York, NY, USA: ACM, 2008, pp. 51–62.

[7] P. Royal, "Alternative medicine: The malware analyst's blue pill," Aug 2008.

[8] T. Garfinkel, K. Adams, A. Warfield, and J. Franklin, "Compatibility is Not Transparency: VMM Detection Myths and Realities," in *Proceedings of the 11th Workshop on Hot Topics in Operating Systems (HotOS-XI)*, May 2007.

[9] "Qwmu open source processor emulator." [Online]. Available: http://wiki.qemu.org/

[10] "Anubis: Analyzing unknown binaries." [Online]. Available: http://anubis.iseclab.org/

[11] R. Paleari, L. Martignoni, G. Fresi Roglia, and D. Bruschi, "A fistful of red-pills: How to automatically generate procedures to detect CPU emulators," in *Proceedings of the 3^{rd} USENIX Workshop on Offensive Technologies (WOOT)*. Montreal, Canada: ACM.

[12] D. Balzarotti, M. Cova, C. Karlberger, C. Kruegel, E. Kirda, and G. Vigna, "Efficient Detection of Split Personalities in Malware," in *Proceedings of the Network and Distributed System Security Symposium (NDSS)*, San Diego, CA, February 2010.

[13] M. G. Kang, H. Yin, S. Hanna, S. McCamant, and D. Song, "Emulating emulation-resistant malware," EECS Department, University of California, Berkeley, Tech. Rep., May 2009.

[14] B. Lau and V. Svajcer, "Measuring virtual machine detection in malware using dsd tracer," *Journal in Computer Virology*, vol. 6, pp. 181–195, 2010, 10.1007/s11416-008-0096-y.

[15] "Juzt-reboot." [Online]. Available: http://www.juzt-reboot.com/

[16] "Partimage." [Online]. Available: http://www.partimage.org/

[17] R. Hund, T. Holz, and F. C. Freiling, "Return-oriented rootkits: bypassing kernel code integrity protection mechanisms," in *Proceedings of the 18th conference on USENIX security symposium*, ser. SSYM'09. Berkeley, CA, USA: USENIX Association, 2009, pp. 383–398.

[18] "Intel®64 and ia-32 architectures software developer's manual." [Online]. Available: http://www.intel.com/Assets/PDF/manual/325384.pdf

[19] "Fast memory copy." [Online]. Available: http://now.cs.berkeley.edu/Td/bcopy.html

[20] X. Jiang, X. Wang, and D. Xu, "Stealthy malware detection and monitoring through vmm-based "out-of-the-box" semantic view reconstruction," *ACM Trans. Inf. Syst. Secur.*, vol. 13, pp. 12:1–12:28, March 2010.

[21] X. Chen, J. Andersen, Z. M. Mao, M. Bailey, and J. Nazario, "Towards an Understanding of Anti-Virtualization and Anti-Debugging Behavior in Modern Malware," in *Proceedings of the 38th Annual IEEE International Conference on Dependable Systems and Networks (DSN '08)*, Anchorage, Alaska, USA, June 2008, pp. 177–186.

[22] N. Xiong, Y. Zhou, H. Liu, and Y. Zhang, "Avmm: Virtualize client with a bare-metal and asymmetric partitioning approach," Submitted, ICC 2011, Tech. Rep., 2011.

[23] A. Depoutovitch and M. Stumm, "Otherworld: giving applications a chance to survive os kernel crashes," in *EuroSys*, 2010, pp. 181–194.

[24] K. Kourai, "Cachemind: Fast performance recovery using a virtual machine monitor," in *Dependable Systems and Networks Workshops (DSN-W), 2010 International Conference on*, 28 2010-july 1 2010, pp. 86–92.

[25] ——, "Fast and correct performance recovery of operating systems using a virtual machine monitor," in *Proceedings of the 7th ACM SIGPLAN/SIGOPS international conference on Virtual execution environments*, ser. VEE '11. New York, NY, USA: ACM, 2011, pp. 99–110.

[26] M. Baker and M. Sullivan, "The recovery box: Using fast recovery to provide high availability in the unix environment," in *In Proceedings USENIX Summer Conference*, 1992, pp. 31–43.

[27] "Norman sandbox analyzer." [Online]. Available: http://www.norman.com/products/sandbox_analyzer/en

Automated Remote Repair for Mobile Malware

Yacin Nadji, Jonathon Giffin, and Patrick Traynor
School of Computer Science, Georgia Institute of Technology
{yacin.nadji, giffin, traynor}@cc.gatech.edu

ABSTRACT

Mobile application markets currently serve as the main line of defense against malicious applications. While marketplace revocations have successfully removed the few overtly malicious applications installed on mobile devices, the anticipated coming flood of mobile malware mandates the need for mechanisms that can respond faster than manual intervention. In this paper, we propose an infrastructure that automatically identifies and responds to malicious mobile applications based on their network behavior. We design and implement a prototype, Airmid, that uses cooperation between in-network sensors and smart devices to identify the provenance of malicious traffic. We then develop sample malicious mobile applications exceeding the capabilities of malware recently discovered in the wild, demonstrate the ease with which they can evade current detection techniques, and then use Airmid to show a range of automated recovery responses ranging from on-device firewalling to application removal.

1. INTRODUCTION

Malware infections on mobile phones have flourished in recent years. Nearly immediately after the introduction of functionality beyond basic telephony, researchers have suggested that such devices were likely targets of malicious software [14, 24]. However, even as mobile operating systems move towards greater sophistication and an increasing number of malicious applications have been discovered, large-scale infection has been avoided. This relative safety can be largely attributed to the revocation capabilities of mobile application markets which, upon discovering or being notified of the existence of a malicious program, can remove it from both the marketplace and installed platforms.

Manually-triggered revocations have been successful because the number of malicious applications has been small. As the rate with which new mobile malware is discovered begins to approach that of traditional malware, estimated by some as greater than 70,000 new samples per day [38], such mechanisms are unlikely to be able to continue to respond rapidly. Specifically, cellular providers will not be able to rely *solely* upon the rapid identification and removal of malware by mobile market operators.

In this paper, we propose, design, and implement Airmid[1], a system for the automated detection of and response to malicious software infections on handheld mobile devices. Our architecture leverages cooperation between network monitors and on-device software components. We place the main detection component within the network so that no malware-specific knowledge needs to be stored at every mobile device. When the network sensor detects malicious traffic (which can be done using traditional security tools including DNS blacklists, domain firewall policies, an IDS, etc), it notifies devices sending or receiving that traffic via an authenticated channel. A protected software agent on each device then identifies executing processes responsible for the malicious traffic and initiates a recovery action to repair the infection: traffic can be filtered at the device, apps can be sandboxed or deleted, or the device itself can be patched or restored to its factory defaults. This system allows malware detection and response to occur at machine speed without human interaction and *without burdening small handheld systems with the computation, storage, and power consumption typical of traditional anti-virus systems.*

As in desktop malware, we expect malicious apps to increase in sophistication as defensive utilities become widespread; however, current malicious applications remain rather simplistic. To facilitate testing with complex malicious apps, we developed laboratory samples of mobile malware exhibiting characteristics common to mature desktop-class malware. The basic functionality of the malware reflects attacks already occuring in cellular devices: our apps leak private data [39], dial premium numbers [26], and participate in botnet activity [36]. To this we added complex evasive functionality: our samples detect the presence of an emulated environment and change their behavior, create hidden background processes, scrub logs, and restart on reboot. These samples demonstrate that mobile device software architectures permit the creation of advanced malware that cannot be easily identified prior to distribution, underscoring the need for rapid recovery after infection. For safety, our malware lacks propagation ability and was tested only on a closed network.

In summary, we make the following contributions:

[1]Airmid is the Celtic goddess of healing, and is known for her ability to bring the dead back to life.

- **Identification of current remediation shortcomings:** While mobile application markets have thus far successfully removed most discovered malware, we argue that such mechanisms are unlikely to remain successful as the *only* mechanism given the expected increase in malicious software for mobile environments. In particular, markets do not have access to all traffic generated by such applications, nor the perfectly analyze all incoming applications prior to distribution.

- **Design and implementation of advanced prototype malware:** We build three proof-of-concept malware instances based on samples collected "in the wild". We then demonstrate the ease with which such malware can resist static and dynamic analysis in a market, making their pre-deployment detection unlikely and the need for a more dynamic "kill switch" mechanism.

- **Cooperatively neutralize malware on infected mobile phones:** We pair a software agent running within a hardened, modified version of the stock Android Linux kernel with a network sensor running Snort. Our network sensor initiates a range of remediation actions that successfully sanitize our advanced prototype malware in under 10 ms. Remediation results can be fed back to mobile markets, allowing both providers and application vendors to more quickly protect customers from malware.

The remainder of this paper is organized as follows: Section 2 compares related research. Section 3 presents an overview of the current state of mobile malware creation and detection, and it demonstrates by example how mobile malware can implement evasive behaviors similar to those of desktop-class malware. Section 4 proposes our remote repair architecture for in-network detection of malware and automatic on-device remediation of the infection. Section 5 presents our specific prototype implementation for Android and an evaluation of the prototype's performance using the evasive malware instances first presented in Section 3. Section 6 includes a discussion of the system, its potential applications, and its current limitations. We conclude in Section 7.

2. RELATED WORK

Mobile malware is becoming increasingly commonplace. While malicious software for these platforms is not new [18–21], the migration towards a small number of mobile operating systems and the increasing power of such devices heightens the probability of large scale infection. Traynor et al. [41] observed that even small numbers of compromised and coordinated phones could be used to cause widespread outages. A number of other authors have built advanced proof-of-concept mobile malware capable of deciphering DTMF tones [37] and collecting video [43]. Mobile malware in the wild has already been found to exfiltrate sensitive data [39], generate calls and text messages to premium numbers [26] and exhibit traditional botnet behaviors [36].

The industrial and research communities have responded with a range of solutions. Numerous anti-virus products have been ported to the most popular mobile platforms [2,9, 17,27]. Others have proposed new platform-based detection systems based on triggers including service thresholds [8], low level API calls [7] and anomalous power usage profiles [28,30]. These device-only approaches fail for a number of reasons, including excessive power requirements for traditional anti-virus system scanning and false positives caused by untrained benign behavior. Tools such as TaintDroid [16] and PiOS [15] can identify information flows in applications, but they require significant manual assistance and do not automatically identify "malicious" behavior; rather, they identify the presence of specific information flows (e.g., IMEIs and IMSIs). Neither approach is currently scalable as a means of analyzing all available mobile applications.

Traffic attribution and provenance are generally difficult problems. IP addresses are well known as weak indicators of the origin of malicious traffic on the Internet. Regular backscatter traffic measurements indicate wide-spread address spoofing [32]. Balasubramaniyan et al. attempt to solve a similar problem in the larger telephony space [5]. Because cellular networks maintain a cryptographic relationship with each user device, attribution of a specific packet to an individual device is possible. However, additional steps must be taken to identify the specific process responsible for traffic. In the context of virtualized desktop systems, Srivastava et al. [40] developed a virtual machine introspection based technique that attributes network traffic to an executing process for the purpose of implementing firewall rules. Our work uses an in-kernel approach to identify a similar correlation between network behaviors and process execution for the purpose of malware infection remediation.

3. MOBILE MALWARE

Malware has begun to move from desktop computing into mobile environments. This section briefly considers mobile malware already deployed, and it then extensively presents highly capable laboratory samples that we created to demonstrate both the evasive technologies available to malware authors on Android platforms and the utility of our proposed remote repair design.

3.1 In the wild

By the first quarter of 2011, over 1000 mobile malware instances have been discovered in the wild, primarily targeting the Symbian OS for feature phones [31]. Android is the second most common target, and the most common smartphone OS victim. Google, the producer of Android and operator of the official app marketplace, has exercised its revocation ability at least three times since 2010 [3,11,12]. The malicious behaviors of Android malware include [31]:

- Privilege escalation to root (DroidDream).
- Bots (Drad.A).
- Data exfiltration (DroidKungFu, SteamyScr.A).
- Backdoor triggered via SMS (Bgyoulu.A)

Similarly, jailbroken devices running Apple iOS experienced a botnet in 2009 [36]. These attacks send and receive traffic across data or voice channels that can be identified by a network intrusion detection system as malicious.

Detection of mobile malware prior to its deployment is as challenging as detection of malware on desktop systems. Centralized app marketplaces provide an opportunity for centralized analysis before the marketplace lists an app. Unfortunately, marketplaces have at least two deficiencies that will enable the proliferation of malicious apps. First, malware authors can write their apps with logic to evade detection or analysis, as we show below with laboratory malware samples. Second, the Android platform allows users to install apps from third-party marketplaces that may make no efforts to verify the safety of the software that they distribute. As a result, we naturally expect malware to successfully reach client devices.

3.2 Enhanced prototype malware

As seen in desktop security, the success of traditional malware detection diminishes as malware incorporates strategies designed to evade detection or analysis. While evasions strengthen the need for post-infection remediation, existing mobile malware samples have not yet added such complexity. To better illustrate the effectiveness and flexibility of Airmid, we created laboratory malware instances suggestive of future malicious developments in the mobile environment. Each instance combines malicious functionality now occurring in real mobile malware with evasive functionality common to desktop malware, and embeds the malicious logic into a benign app. Inserting malware into a benign app and rehosting the app on a third-party marketplace has become a common malware distribution strategy. For safety, we created our laboratory malware samples without distribution or propagation functionality.

We created three laboratory mobile malware samples: a Twitter client that leaks private data ("Loudmouth"), a Facebook client sync app that dials premium numbers ("2Faced"), and a mobile bot ("Thor"). Each piece of malware is actually a maliciously modified open-source Android application. Our malware *requires no additional permissions* from the unmodified applications, which are listed in Table 1. All samples include evasive techniques and exceed the capabilities of mobile malware currently in the wild.

3.2.1 Loudmouth

Our first malware instance, Loudmouth, combines:

- Malicious mobile functionality: *Data exfiltration.*
- Evasive functionality: *Malware analysis environment detection.*
- Benign host app: *Twitter client.*

The principal malicious functionality of Loudmouth is data exfiltration, a powerful attack in a mobile context. As users embrace the functionality of smartphones, they place significant personal data on their devices. Even basic feature phones store contact lists, messages, and call databases. GPS-enabled devices can leak location information, turning a mobile device into a tracking system. Such data is useful for criminal activities.

Loudmouth appears to the user as a benign Twitter client. We augmented the Nanotweeter [34] codebase with just 143 lines of new, malicious code. The app provides the expected usual functionality, but it also leaks private information to a server owned by the attacker (here, us) during use. It steals Twitter account credentials, the phonebook database, and the most recently available GPS location. When active, Loudmouth creates malicious network traffic that contains private data and is transmitted to the malware author's server.

Desktop-class malware is commonly analyzed by executing the malware within a virtualized or emulated environment. Malware authors have hence begun to include checks for analysis environments, and the malware alters its behavior so that it does not exhibit malicious functionality during analysis [13, 22]. Mobile malware does not yet contain such sophistication; however, should application marketplaces begin to verify apps via runtime analysis, then mobile malware authors may begin to include checks for analysis environments. We implemented multiple checks in Loudmouth.

To thwart dynamic analysis, Loudmouth contains several checks to *determine its environment*, and it hides all malicious behavior when run on an emulator or a developer phone. The Android platform provides straightforward methods of detection. Loudmouth first queries the "brand", "device", and "model" Java environment variables and compares their values against a whitelist of known consumer phones. The malware then checks for the Radio Interface Layer (RIL) library, the Google Maps application, and any DRM system components. These three components are either supplied by hardware device manufacturers or are proprietary, and hence none are distributed with the emulator or with developer phones. These restrictions are actively enforced even for third-party distributions of the operating system [42]. Loudmouth can thus determine whether the phone is emulated or a physical device, as well as specific details of the phone. To the best of our knowledge, we are the first to investigate and implement these techniques in mobile phones.

3.2.2 2Faced

Our second malware instance, 2Faced, combines:

- Malicious mobile functionality: *Premium number dialer.*
- Evasive functionalities: *Log sanitization* and a *hidden native process.*
- Benign host app: *Facebook sync.*

Premium number dialer applications are examples of historic malware that once targeted dial-up Internet users. These attacks resulted in revenue for the owners of the premium numbers, as users had little chance of refuting their expensive phone bills [10]. These attacks have recently seen a resurgence on mobile devices [26].

2Faced is a new generation of the premium number dialer. It is a Facebook contact sync app that dials premium numbers late at night when the mobile device is not in use. The attacker can generate revenue until the user of the device notices these hidden calls. It creates detectable, malicious telecom traffic to the premium numbers.

Our malware instance includes two evasive actions designed to prolong its existence on infected devices. First, it evades early detection from a vigilant user by removing its entries from the device's call logs. Even observant users will be unable to detect malicious activity, maximizing both the financial loss for the user and the gain for the attacker. Second, the actual malicious dialer functionality is hidden in an ARM executable (2facedsrv) installed by 2Faced into the device's local storage. We evade detection via static app analysis by distributing 2facedsrv inside the image resources of 2Faced, though the executable could also be retrieved remotely. The native process is only 14 lines of code, and its

Table 1: Summary of permissions requested by sample mobile malware.

Permissions	Loudmouth	2Faced	Thor
Network access	✓	✓	✓
Coarse location	✓		✓
GPS location	✓		
Read contacts		✓	
Write contacts		✓	
Call phone		✓	

extraction and invocation from 2Faced requires fewer than 100 lines of code in the Java-based portion of 2Faced. This delivery method has recently been observed in the iKee.B iPhone botnet [36].

Native processes are particularly useful to a malware author. Here, 2facedsrv is able to execute its malicious behavior because it is launched with the same permissions as the Java-based parent 2Faced. Notably, it is not terminated when the user exits the parent application, but rather runs until the device reboots. After a reboot, it then restarts when 2Faced is relaunched. Native ARM processes are not listed in Android's default interface, so only the most observant and expert users might detect it by reviewing a full process list.

3.2.3 Thor

Our third and final malware instance, Thor, combines:

- Malicious mobile functionality: *Bot client.*
- Evasive functionality: *Persistence across reboot.*
- Benign host app: *Weather display.*

Our last malware sample demonstrates how current desktop malware can become equally dangerous in mobile environments. We ported to Android an existing Windows-based botnet client [25], which we refer to as Thor (1,353 lines of code). Its original functionality included system command execution, local file access, and denial-of-service (DoS) attacks using HTTP, UDP, and TCP SYN floods. For safety reasons, our port removed its propagation functionality and pointed its command-and-control (C&C) connection to machines under our control. We added the bot client abilities to MyWeather [33], an open-source weather application. When executing, it generates a malicious network footprint containing both C&C and attack traffic.

Unlike the previous malware instances, Thor is installed as a service separate from MyWeather. The service is immediately active and restarts on every boot of the device. It first connects to two different predefined IRC channels used as the main control channel for the botnet. After connecting, Thor waits for one of our commands, including data exfiltration or DoS.

Mobile bot clients can be used for targeted attacks against the cellular network. By using the GPS location, the botnet can target specific base stations or regions for attack. Bot clients with additional capabilities can potentially augment these location-based denial-of-service attacks with both voice and text message traffic, making attacks more effective. In large numbers, such compromised devices can bring down large portions of the cellular network [41].

4. ARCHITECTURE

We expect that malware such as that presented in Section 3 will ultimately be successfully installed on unwitting mobile devices regardless of protective measures taken to prevent their spread. Automated remediation provides opportunities to repair infected systems. Here, we describe the design of our proposed remote repair system, Airmid, which implements the detection, attribution, and remediation of malware infections in mobile environments.

4.1 Threat model & design principles

Airmid detects and responds to malicious software execution on cellular and mobile devices. This is a straightforward threat model encompassing two primary types of attack. First, we expect attackers to successfully install malware on mobile devices via a variety of usual mechanisms, such as drive-by downloads or automated propagation, and by mobile-specific techniques, including distribution on official and third-party app marketplaces. Second, attackers can subvert the correct execution of a benign app by exploiting a security defect in the app's design or implementation. Both attacks result in undesirable software execution on the targeted device, possibly with root access.

In this threat model, the correct functioning of our proposed prototype system then depends on several reasonable assumptions holding true. We assume the existence of the following:

- A protected software layer on the device lower than the level at which the malware executes. Our remediation design includes on-device software that, if subverted, will result in unsuccessful recovery. Reasonable candidate layers include the kernel (if kernel-level malware can be prevented) and a hypervisor (if virtualized environments can be created on a mobile device). For our prototype implementation, we chose to operate at the kernel level and harden the kernel appropriately.

- A communication channel between the network and each device, even if the channel is intermittently connected. Our cooperative design sends signals from network detection systems to software agents on infected systems, and we expect that these signals will ultimately reach any device. While the permanent absence of a communication channel prevents remediation, it likewise prevents malice from spreading beyond the device.

- Detectable malicious behavior in the network. We chose to use network-based attack detection rather than on-device detection to avoid the costly process of traditional signature-based malware detection. In mobile environments, the storage, computation, and power costs of malware detection may be prohibitive, and moving detection into the network avoids those costs altogether.

We have no ability to prevent device owners from rooting or jailbreaking their devices and then overwriting the

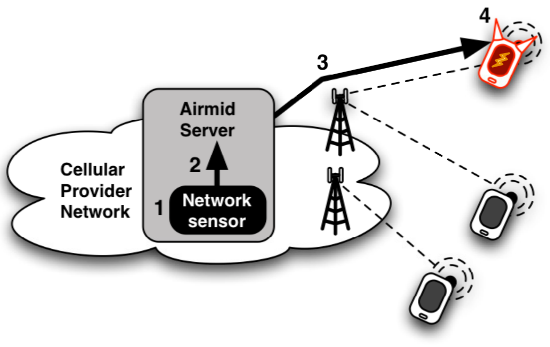


Figure 1: High-level workflow of automated remote repair. (1) An infected device sends or receives malicious data or voice communication detected by network sensors deployed within a provider's network. (2) The provider notifies an Airmid server of the offending traffic. (3) The server transmits an authenticated message to the protected client-side Airmid software on the infected device. (4) The software attributes the malicious network traffic to a responsible process, and initiates remediation actions.

Airmid software. Reflashed devices simply will not benefit from automated repair of infections and may simply be denied service by their provider should malicious traffic be detected (as currently happens).

4.2 Remote repair

Our remote repair design consists of a server-based infection detection system and an on-device attribution and remediation system that cooperate to automatically identify and repair infected devices. The network-based detection system has a broad view of a provider's entire network: it can observe behaviors correlated across multiple devices that would not be meaningful to any single device, such as a distributed denial-of-service attack or exfiltrated data all transmitted to the same server. The on-device software permits detailed, real-time inspection of device state and the ability to effect changes that restore benign operation. The server automatically initiates remediation of infected devices via an authenticated communication channel. Airmid's cooperative design allows these strengths to be combined: broad network perspective, detailed device knowledge, and effective attack response.

During an active infection, Airmid operates as shown in Figure 1. Depending upon the nature of the specific infection, an infected device sends and receives malicious data—such as propagation traffic, command-and-control, and exfiltrated data—and generates calls or text messages to high-cost recipients. A network sensor (which could be using data sources including DNS blacklists, patterns of known botnets [23] or other technique) monitoring data and telecom networks (1) identifies infected devices based upon these malicious traffic patterns. It notifies (2) the Airmid server of the offending devices and traffic, and the server sends the alerts across the authenticated channel to each infected device (3). The on-device software receives the alert (4), inspects the kernel's internal state to attribute the malicious network traffic to a particular process or service, and then initiates one or more remediation actions against that process.

The Airmid on-device software executes from within a trusted, low layer on the device, such as a hardened kernel or hypervisor. It has two responsibilities: attribute traffic to malware, and repair the infection. Attribution is closely tied to a particular mobile operating system's implementation, so we defer the presentation of our Android-specific analysis to Section 4. Once it has identified software on a device responsible for whatever network behaviors triggered the detection system, the software executes repair actions to disable malicious activity or to remove malware entirely. We employ the following recovery options, loosely ordered from least likely to create side-effects to most likely:

- Process termination.
- On-device traffic filtering.
- App update.
- Device update.
- File removal.
- Factory reset.

Decision logic within the software component will select an action based upon the nature of the offending network traffic, the identity of the attributed process, and that process' privilege level.

4.3 Authenticated communication

The Airmid software executes attribution and recovery when signaled by the network intrusion detection system. The communication channel carrying these signals clearly must be secured to prevent illicit triggering of operations that change a device's state. Airmid authentication augments initial UMTS authentication in a cellular network, as shown in Figure 2. It extends the scope of authentication from the device and network to the Airmid device and server components, reusing existing network functionality and secret keys. This has three benefits: first, no additional secrets need to be shared or updated for this authentication, as one private key manages both authentication types. Second, the difficulty of integration into existing networks

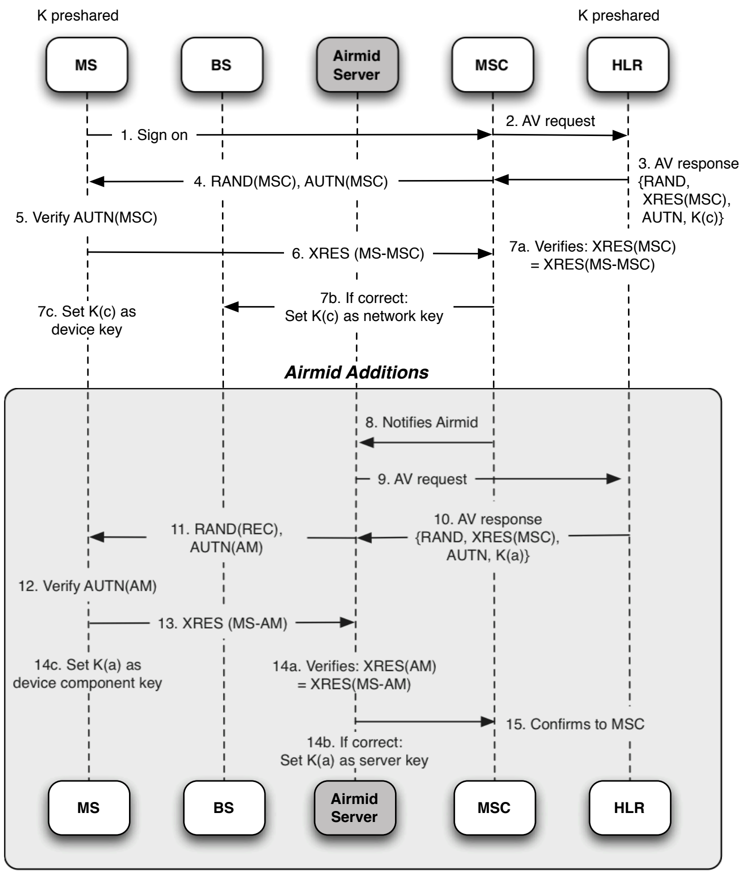

K preshared
K preshared
MS
BS
Airmid Server
MSC
HLR
1. Sign on
2. AV request
3. AV response {RAND, XRES(MSC), AUTN, K(c)}
4. RAND(MSC), AUTN(MSC)
5. Verify AUTN(MSC)
6. XRES (MS-MSC)
7a. Verifies: XRES(MSC) = XRES(MS-MSC)
7b. If correct: Set K(c) as network key
7c. Set K(c) as device key
Airmid Additions
8. Notifies Airmid
9. AV request
10. AV response {RAND, XRES(MSC), AUTN, K(a)}
11. RAND(REC), AUTN(AM)
12. Verify AUTN(AM)
13. XRES (MS-AM)
14a. Verifies: XRES(AM) = XRES(MS-AM)
14b. If correct: Set K(a) as server key
14c. Set K(a) as device component key
15. Confirms to MSC
MS
BS
Airmid Server
MSC
HLR

Figure 2: Authentication flow: the Airmid authentication extends existing UMTS authentication. The device and network are mutually authenticated after UMTS authentication and all further traffic is encrypted using a session key. Airmid extends this authentication process to create a secure channel between the Airmid server and the process running on the mobile phone.

is reduced, as no additional authentication servers are required. Third, the approach provides the same guarantees as existing UMTS authentication and can be upgraded at the same time if needed.

The final step of UMTS authentication completes the mutual authentication and sets a session key used for encryption. Airmid extends this authentication process.[2] All traffic is encrypted through the existing session key, which guarantees a secure connection through the cellular network. At this point the Mobile Switching Center (MSC) notifies the Airmid server of the newly authenticated device. The server then requests UMTS Authentication Vectors (AV) from the Home Location Register (HLR), the device storing the long-term credentials for traditional telephony authentication. These AVs consist of the RAND, XRES and AUTN values, which are the randomly generated user challenge, expected user response, and the authentication token respectively. The Airmid server then forwards its values of RAND and AUTN to the device. Through these values the device component verifies the server's knowledge of the secret key and generates its response based on its key. This response allows the Airmid server to verify that the device is aware of the secret key, hence, completing mutual authentication. If the server or device are unable to prove their knowledge of the shared secret, then the authentication fails. The session key $K(a)$, which is never sent over the air, is then used for encryption. Cryptographic algorithms used in UMTS are similarly used for Airmid authentication and encryption.

[2]We encourage readers interested in more information about UMTS authentication procedures to read either the standards document [1] or Traynor et al. [41].

5. IMPLEMENTATION

Our prototype implementation is separated into two components: passive network analysis and signaling within the network, and remediation logic that responds to these signals on the device. To emulate sensors within the cellular network, we built a packet sniffer that analyzes network traffic from an HTC Dream mobile phone running Android 1.6. A secure communication channel is established between the sensor and the kernel when a sensor needs to issue remediation commands to an infected device, whereupon the device performs various actions to remediate the infection.

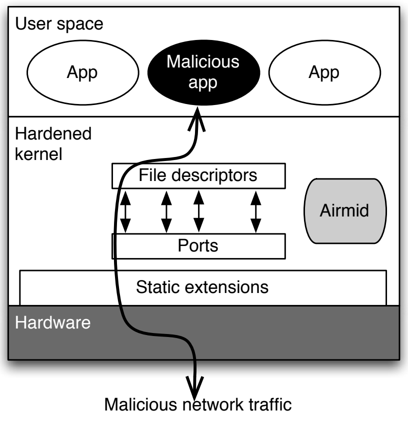


Figure 3: Airmid's on-device component resides within a hardened Android kernel. In particular, all kernel extensions are statically linked into the kernel so that its loadable module support can be disabled.

5.1 Network component

Our prototype network component includes both an attack detection system and the Airmid server communicating with infected devices. For detection, we used Snort, an off-the-shelf network intrusion detection system. We wrote our Airmid server in Python using the packet creation library Scapy [6]. As a mobile device communicates with foreign network assets, Snort analyzes incoming and outgoing traffic. When it flags traffic as malicious, it notifies the Airmid server which subsequently initiates authenticated two-way communication with the suspect device. It sends to the device a remediation message that provides the device's port number from the attack alert. This information is used by the Airmid on-device software to determine which processes and applications are acting maliciously. The device's software decides which remediation strategies are appropriate and relays how the infection was remediated to the sensor.

5.2 Device component

Our threat model considers a device's kernel to be the trusted computing base, so provenance and remediation actions run in kernel-mode. We modified the Linux kernel version 2.6.29 to include our prototype implementation. We hardened the kernel against direct compromise by disabling its ability to dynamically load kernel modules. Our modifications required approximately 1,200 lines of C. When the device is booted, a separate kernel thread is launched to perform infection provenance and remediation. When a message is received from an authenticated Airmid server, the device: determines provenance of the offending traffic, determines and enacts a remediation strategy, and relays the device's decision to the network sensor. We now discuss the implementation of the provenance mechanism that leverages Android's existing application sandboxing techniques as well as implementation details for the remediation strategies. Figure 3 offers a high-level overview of this component.

5.2.1 *Infection provenance*

When a user installs an application on a mobile device, Android generates a unique Linux user ID that persists for the application's lifetime [4]. All files owned by the application are also assigned this user ID. Given a user ID, we can find the Android package file (`.apk`) of the application as well as configuration files and native binaries dropped by the application both at install-time and runtime.

When a device receives a remediation signal from a network sensor, the device crawls two kernel data structures: `inet_hashinfo`, which contains port usage information, and `task_struct`, which contains process information. While `inet_hashinfo` contains port information only for TCP connections, the technique itself is general and could be extended to perform provenance on UDP-based traffic. The port number sent to the device by the network sensor is mapped to an open socket by iterating through `inet_hashinfo`. We then look for this socket in the list of `task_struct`s to find the process that holds said socket. A visualization of this process is shown in Figure 4. From the `task_struct`, we then extract the user ID, u, of the malicious process and crawl the disk to find all files owned by this application.

5.2.2 *Remediation strategies*

Depending on the value of u, the user ID identified in the previous step, the Airmid software initiates one or more of the following repair actions: creation of kernel-protected local firewall rules to block the malicious traffic, termination of processes running under u, removal of the application package (apk) owned by u, and removal of all files owned by u. If $u < 10000$, then it is a system user ID corresponding to a core service whose benign functionality may have been subverted. Airmid will only block the malicious traffic by creating appropriate firewall rules. If $u \geq 10000$, then u is an application user ID. Airmid will terminate all processes running with ID u and will remove the application package owned by u. We identify applications for removal by parsing the file containing all installed application packages and the user IDs given to them Android at install-time (`/data/system/packages.xml`). Finally, Airmid scans the list of running processes to see if any native ARM processes are executing with user ID u. If so, it scans the full storage of the device to purge all files owned by u.

To firewall malicious traffic, the client interfaces with the Android kernel's netfilter [35] hooks to provide lightweight packet filtering. We add rules that prohibit traffic destined

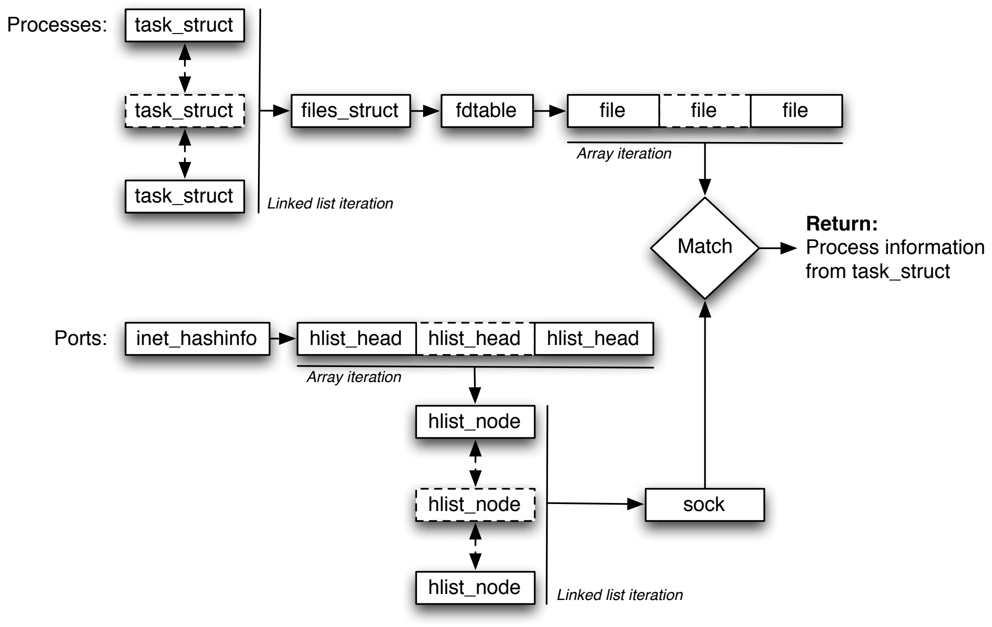


Figure 4: Traversal of data structures to perform malicious traffic provenance of the mobile device.

Table 2: Average Command Execution Time (95% confidence interval)

Command	Mean (μs)	C.I. (μs)
Factory Reset	1504	± 38
Device Status (Process List)	2991	± 48
Process Termination	6537	± 1423
File Removal	6115	± 31
Device-Side Filtering Rule	7149	± 826
Application Removal	9174	± 1123

for the IP addresses actively in use by processes owned by u. Additional provenance checks determine all active IP addresses if more than one process is running under u. To prevent observation or alteration of rules from userland, Airmid does not register its rules with `ip_tables` but rather maintains its own shadow `ip_tables` data structure.

5.3 Performance evaluation

We characterize the overheads associated with the Airmid architecture through a performance analysis. We measured each operation 100 times and provide 95% confidence intervals.

We measured each of the operations discussed in Section 4.2 and recorded very modest overheads for all proposed functions. Table 2 summarizes our experiments. While seemingly surprising, the factory reset function takes the least time with an average of 1504 μs ($\pm 38\mu$s). However, a factory reset can be performed simply by deleting a single configuration file (`data.img`). The most expensive operation, application removal, requires the most time with an average of 9174 μs ($\pm 1123\mu$s). This result was also somewhat expected as it includes the costs of both process termination and deletion of multiple files.

These results demonstrate the lightweight nature of Airmid. Unlike more traditional anti-virus architectures, which require regular scanning of all of the contents of the phone (an operation taking on the order of 10s of seconds), our approach attributes malicious behavior directly to an application and allows a targeted response to be implemented. This makes our approach more conscious of the power constraints associated with mobile devices.

6. DISCUSSION

The concept of automating remote infection identification and local device repair raises natural questions over the organizational control of remediation and the security of the required on-device software.

6.1 Airmid control

Airmid provides a powerful architecture to respond to the growing problem of mobile malware. While we believe that this approach can help the vast majority of individual users, we recognize that there exist numerous parties that may not trust a cellular network provider to perform these operations. For instance, phones used by members of government agencies or employees of a rival company may not wish to outsource their malware remediation. *We do not intend Airmid to be a "one size fits all" solution* and discuss how such systems can be deployed in reality.

Our architecture is general enough such that the functions described in this paper can be implemented by a separate entity or cloud service. Many companies already require data traffic from their corporate phones to be proxied via VPN through their network.[3] Such entities could simply deploy

[3]We note that the Blackberry BES provides VPN service and offers to limit the applications allowed to run on mobile devices. Our architecture is more flexible and does not derive its protections from the explicit specification of ap-

an Airmid-speaking service in their network and then configure their devices (e.g., by changing keying material on the device) to only accept commands from that server. Individuals not wanting such control could similarly alert the provider, perhaps through an out-of-band resource such as a web interface or when the device/service plan is purchased. We leave the debate over opt-in/opt-out strategies to the policy community, but note that a range of economic incentives could be provided to sway customers towards the use of Airmid regardless of the chosen approach.

Airmid does not tie a device's security to arbitrary cellular providers. For instance, when traveling abroad, mobile phones running the modified Airmid kernel would be able to receive roaming telephony and data service as normal without providing their "visited" network with control over Airmid's functionality. We believe that this approach is necessary as laws potentially protecting customers from abuse by their provider domestically are unlikely to apply internationally. While this means that misbehaving roaming devices are more likely to simply be denied service by their visited network (as is currently the practice), we believe that this tradeoff is ultimately more secure.

Lastly, the guarantees provided by Airmid are only as good as the detection mechanisms used by the network sensor. For instance, policies enacted against traffic from or destined for known malicious domains or bots are likely to be effective. However, a system relying on an IDS reporting any "anomalous" traffic will produce false positives. Accordingly, these mechanisms must be selected carefully. We leave this selection to future work, where we intend to observe and further characterize the network traffic of large amounts of mobile malware.

6.2 Device hardening

The correct operation of our architecture requires that the device is able to protect the Airmid software. We chose to do this in our proof of concept implementation by hardening the stock Android Linux kernel through steps including disabling dynamic loading of kernel modules. Given that a significant proportion of traditional malware abuses this mechanism and that mobile devices generally do not take advantage of this capability, we believe that this is a first strong step in preventing kernel compromise. However, we recognize that deploying Airmid on real systems may require additional hardening. An increasingly popular approach is the use of virtualization. While a number of virtualization solutions are evolving for mobile devices [29], we believe that such mechanisms are relatively expensive and may themselves not yet be ready for widespread deployment. As our threat model indicates, it is necessary to identify a protection layer with reasonable tradeoffs in which Airmid can run. We believe that our approach is reasonable given current mechanisms, but note that best practices for device hardening are still an active research area.

7. CONCLUSION

As mobile devices begin to see an increasing volume of malicious applications, the ability of application markets to identify and remove such applications in a timely manner will be lost. We respond to this problem by developing Airmid, an automated system for the remote remediation of mobile malware. Upon the detection of malicious traffic, the cellular network interacts directly with the source device to identify the provenance of that traffic. The device can then perform remediation ranging from filtering the offending traffic to uninstalling the application. We then demonstrate that Airmid has very low overhead. In so doing, we demonstrate that the detection and removal of malicious applications can scalably be outsourced to cellular providers and applications markets, ultimately providing faster responses to infection.

plications.

Acknowledgments

We would like to thank Michael Iannacone, Ferdinand Schober and our anonymous reviewers for their helpful comments in the completion of this paper. This work was supported in part by the US National Science Foundation (CNS-0916047). Any opinions, findings, conclusions or recommendations expressed in this publication are those of the authors and do not necessarily reflect the views of the National Science Foundation.

8. REFERENCES

[1] 3rd Generation Partnership Project. General packet radio service (GPRS). Technical Report 3GPP TS 23.060 v8.0.0.

[2] Airscanner AntiVirus for Windows Mobile, Accessed 2011. `http://www.airscanner.com/downloads/av/av.html`.

[3] C. Albanesius. Google pulls malware-infected apps from Android market. *PCmag.com*, June 2011.

[4] Android Developers. Security and permissions, Accessed 2011. `http://developer.android.com/guide/topics/security/security.html`.

[5] V. Balasubramaniyan, A. Poonawalla, M. Ahamad, M. Hunter, and P. Traynor. PinDr0p: Using Single-Ended Audio Features to Determine Call Provenance. In *Proceedings of the ACM Conference on Computer and Communications Security (CCS)*, 2010.

[6] P. Biondi. Scapy. `http://www.secdev.org/projects/scapy/`, Accessed 2011.

[7] A. Bose, X. Hu, K. G. Shin, and T. Park. Behavioral detection of malware on mobile handsets. In *Proceedings of the International Conference on Mobile Systems, Applications, and Services (MobiSys)*, June 2008.

[8] A. Bose and K. Shin. Proactive security for mobile messaging networks. In *Proceedings of the ACM Workshop on Wireless Security (WiSe)*, Sept. 2006.

[9] Bullguard mobile antivirus, Accessed 2011. `http://www.bullguard.com/why/bullguard-mobile-antivirus.aspx`.

[10] R. E. Calem. Scam costs net users thousands in transatlantic telephone bills. *New York Times*, Feb. 11, 1997.

[11] R. Cannings. Exercising our remote application removal feature. Android developers blog, June 2010. `http://android-developers.blogspot.com/2010/06/exercising-our-remote-application.html`.

[12] R. Cannings. An update on Android market security. Google Mobile blog, Mar. 2011. `http://googlemobile.blogspot.com/2011/03/update-on-android-market-security.html`.

[13] X. Chen, J. Andersen, Z. Mao, M. Bailey, and J. Nazario. Towards an understanding of anti-virtualization and anti-debugging behavior in modern malware. In *Proceedings of the IEEE International Conference on Dependable Systems and Networks (DSN)*, June 2008.

[14] D. Dagon, T. Martin, and T. Starner. Mobile phones as computing devices: The viruses are coming! *IEEE Pervasive Computing*, 3(4):11–15, 2004.

[15] M. Egele, C. Kruegel, E. Kirda, and G. Vigna. PiOS: Detecting privacy leaks in iOS applications. In *Proceedings of the ISOC Networking & Distributed Systems Security Symposium (NDSS)*, Feb. 2011.

[16] W. Enck, P. Gilbert, B.-G. Chun, L. P. Cox, J. Jung, P. McDaniel, and A. N. Sheth. TaintDroid: An information-flow tracking system for realtime privacy monitoring on smartphones. In *Proceedings of the USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, Oct. 2010.

[17] F-Secure mobile security, Accessed 2011. `http://www.f-secure.com/en_US/products/mobile/mobile-security/`.

[18] F-Secure Corporation. F-Secure computer virus descriptions: Cabir, Dec. 2004. `http://www.f-secure.com/v-descs/cabir.shtml`.

[19] F-Secure Corporation. F-Secure computer virus descriptions: Mabir.A, Apr. 2005. `http://www.f-secure.com/v-descs/mabir.shtml`.

[20] F-Secure Corporation. F-Secure computer virus descriptions: Skulls.A, Jan. 2005. `http://www.f-secure.com/v-descs/skulls.shtml`.

[21] F-Secure Corporation. F-Secure malware information pages: Worm:SymbOS/Commwarrior, 2008. `http://www.f-secure.com/v-descs/commwarrior.shtml`.

[22] T. Garfinkel, K. Adams, A. Warfield, and J. Franklin. Compatibility is not transparency: VMM detection myths and realities. In *Proceedings of the Workshop on Hot Topics in Operating Systems (HotOS)*, May 2007.

[23] G. Gu, P. Porras, V. Yegneswaran, M. Fong, and W. Lee. BotHunter: Detecting malware infection through IDS-driven dialog correlation. In *Proceedings of the 16th USENIX Security Symposium*, Aug. 2007.

[24] C. Guo, H. J. Wang, and W. Zhu. Smart phone attacks and defenses. In *Proceedings of Third ACM Workshop on Hot Topics in Networks (HotNets)*, Nov. 2004.

[25] Hack Forums. Windows botnet source, Accessed 2011. `http://www.hackforums.net/showthread.php?tid=108411`.

[26] InfoSecurity.com. Premium rate calling Android malware spotted in the wild, May 2011. `http://www.infosecurity-us.com/view/18301/premium-rate-calling-android-malware-spotted-in-the-wild/`.

[27] Kaspersky mobile security, Accessed 2011. `http://www.kaspersky.com/mobile_downloads`.

[28] H. Kim, J. Smith, and K. Shin. Detecting energy-greedy anomalies and mobile malware variants. In *Proceedings of the International Conference on Mobile Systems, Applications, and Services (MobiSys)*, June 2008.

[29] L4Android - Android on top of L4, Accessed 2011. `http://l4android.org/`.

[30] L. Liu, G. Yan, X. Zhang, and S. Chen. Virusmeter: Preventing your cellphone from spies (sic). In *Proceedings of the Symposium on Recent Advances in Intrusion Detection (RAID)*, Sept. 2009.

[31] McAfee Labs. McAfee threats report: First quarter 2011, June 2011.

[32] D. Moore, G. M. Voelker, and S. Savage. Inferring Internet denial-of-service activity. In *Proceedings of the USENIX Security Symposium*, Aug. 2001.

[33] My Weather, Accessed 2011. `http://island.byu.edu/unclass/content/android-web-service-app-my-weather-running-and-full-source-code`.

[34] Nanotweeter, Accessed 2011. `http://code.google.com/p/nanotweeter/`.

[35] Netfilter/iptables project, Accessed 2011. `http://netfilter.org/`.

[36] P. Porras, H. Saidi, and V. Yegneswaran. An analysis of the iKee.B (Duh) iPhone botnet. Technical report, SRI International, Dec. 2009.

[37] R. Schlegel, K. Zhang, X. Zhou, M. Intwala, A. Kapadia, and X. Wang. Soundcomber: A stealthy and context-aware sound trojan for smartphones. In *Proceedings of the ISOC Network and Distributed Systems Security (NDSS) Symposium*, Feb. 2011.

[38] SecurityWeek News. New malware jumps to 73,000 samples every day, says PandaLabs, Mar. 2011. `http://www.securityweek.com/new-malware-jumps-73000-samples-every-day-says-pandalabs`.

[39] M. Shipman. Enter the hacker: New DroidKungFu malware is bad news for Androids. The Abstract blog, June 2011. `http://web.ncsu.edu/abstract/technology/wms-droidkungfu/`.

[40] A. Srivastava and J. Giffin. Tamper-resistant, application-aware blocking of malicious network flows. In *Recent Advances in Intrusion Detection (RAID)*, Cambridge, Massachusetts, Sept. 2008.

[41] P. Traynor, M. Lin, M. Ongtang, V. Rao, T. Jaeger, T. La Porta, and P. McDaniel. On cellular botnets: Measuring the impact of malicious devices on a cellular network core. In *Proceedings of the ACM Conference on Computer and Communications Security (CCS)*, Nov. 2009.

[42] T. Wimberly. Cyanogenmod in trouble? Android and Me blog, Sept. 2009. `http://androidandme.com/2009/09/hacks/cyanogenmod-in-trouble/`.

[43] N. Xu, F. Zhang, Y. Luo, W. Jia, D. Xuan, and J. Teng. Stealthy video capturer: A new video-based spyware in 3G smartphones. In *Proceedings of the ACM Conference on Wireless Network Security (WiSec)*, Mar. 2009.