HoneyC - The Low-Interaction Client Honeypot

Christian Seifert, Ian Welch, Peter Komisarczuk
{cseifert, ian.welch, peter.komisarczuk}@mcs.vuw.ac.nz

August 2006

1 Introduction

A honeypot is a security device that is designed to lure malicious activity to itself. Capturing such malicious activity allows for studying it to understand the operations and motivation of attackers, and subsequently helps to better secure computers and networks. A honeypot does not have any production value. "It's a security resource whose value lies in being probed, attacked, or compromised" [18]. Because it does not have any production value, any new activities or network traffic that comes from the honeypot indicates that it has been successfully compromised. As such, a compromise is very easy to detect on honeypots. False positives, as commonly found on traditional intrusion detection systems, do not exist on honeypots.

Honeypots' origins can be traced far back to military concepts and usage, but first appeared in the area of computer security in the 1980s. Stoll describes the hunt of a hacker in 1986 [19]. In order to monitor the intruder on a live system, Stoll and his colleagues provided "bait", fake military reports, to lure the attacker into a particular area of their system. While this was not the honeypot that we know today, it was the first attempt of "catching flies with honey". The initial honeypot that made use of a simulated environment was described by Cheswick in his account of tracking the Dutch hacker Berferd in 1991 [5]. In the late 90s, attempts to lure and observe attackers moved to the mainstream with the introduction of various tools [6, 1, 4] and commercial products [3, 2].

In our taxonomy of honeypots [17], we classified the majority of these systems according to the taxonomy's developed classification scheme. The main class identified of honeypots was the interaction level. Possible values of the interaction level are high and low. The high interaction level denotes that the honeypot system allows for full functional interaction. An example of such a honeypot is the Honeynet [4]. A low interaction level signifies that the functionality is limited, for example by using emulated services. This strategy is followed by Honeyd [14].

Pouget et al compared the interaction levels [13] and concluded they are complementary in nature and allow for more accuracy, depending on the circumstances of deployment and goals of data collection. For example, it might be unnecessary to deploy a high interaction honeypot in on a global scale as global data is likely to be similar; low interaction honeypots are more suited for this situation. On the other hand, low interaction honeypots are not suited for an in-depth investigation of attackers actions once a honey-

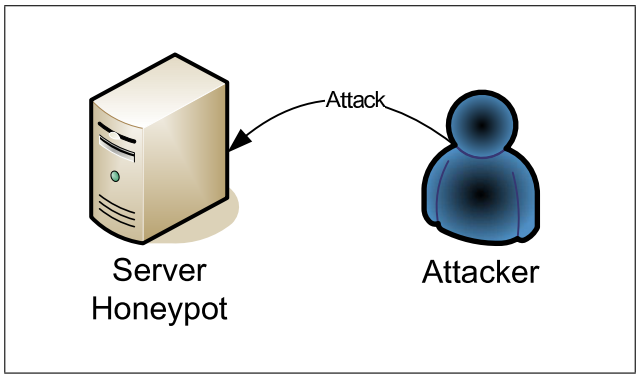

Figure 1: Server Honeypot Architecture

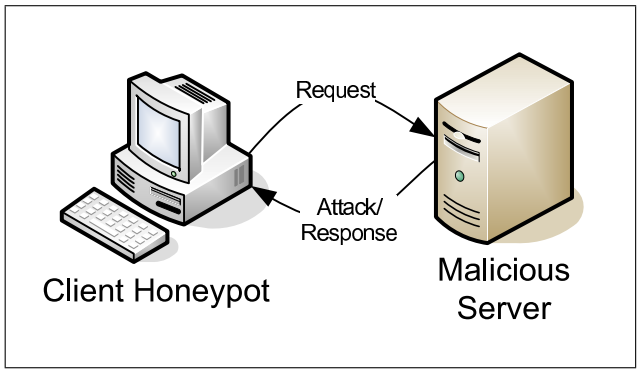

Figure 2: Client Honeypot Architecture

pot has been successfully compromised. High interaction honeypots are required to meet these goals as they expose the full functional spectrum of a computer system for the attacker to interact with and therefore allow for collection of the desired data.

Typically when discussing honeypots we reference *server* honeypots, such as the ones mentioned above that expose server services and wait to be attacked as shown in figure 1. The focus of this paper, however, is a newer technology called *client* honeypots that deals with a different attack vector. First, in section 2 we introduce client honeypots and review existing client honeypot technology. In section 3 we introduce the notion of low interaction client honeypots and introduce the first known implementation of such a tool, called HoneyC.

2 Traditional Client Honeypots

Client honeypots are necessary because they are able to detect an attack vector that server honeypots are not able to detect. These so-called client side attacks are assaults of clients that originate from malicious servers. This could be a seemingly harmless visit to a website with a browser. As part of a server's response to a client request, the malicious website might serve code that is targeted at exploiting a vulnerability of the browser as shown in figure 2. As a result, a mere visit to the website might leave a machine exploited with malware. Client honeypots are designed to interact with servers and detect the attacks of servers.

The idea of client honeypots was formally articulated in June 2004 by honeypot pioneer Lance Spitzner. Fewer than a handful of client honeypots exist today: Honeyclient [20]; Honeymonkey [21]; and the client honeypot of the University of Washington (UW) [12]. These client honeypots focus on malicious web servers, which they interact with by driving a web browser on the honeypot system. Honeyclient detects successful attacks by monitoring changes to a list of files, directories, and system configuration after the Honeyclient has interacted with a server. Honeymonkey also detects intrusions by monitoring changes to a list of executable files and registry entries, but Honeymonkey goes a step further by adding monitoring of the child processes to its repertoire to detect client side at-

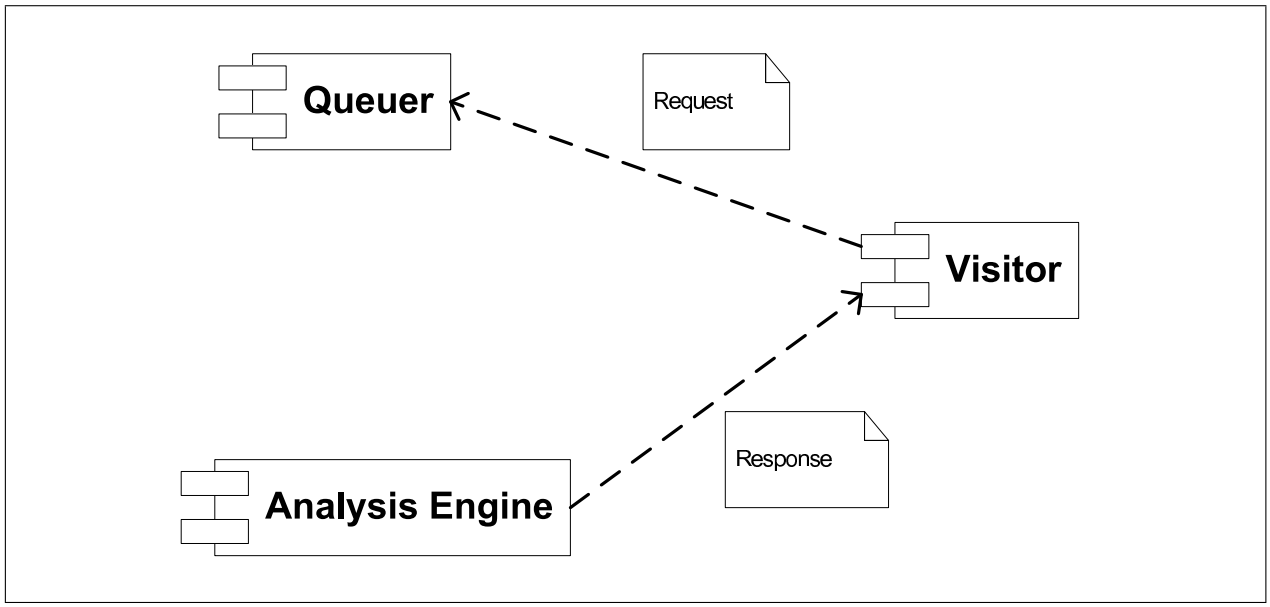

Figure 3: HoneyC Component Diagram

tacks. The UW client honeypot uses event triggers of file system activity, process creation, registry activity, and browser crashes to identify client side attacks.

All these client honeypots can be classified as high interaction client honeypots because they make use of a real browser within a real operating system environment and monitor the state of the entire system. With this interaction level come a few disadvantages. First, operation of high interaction client honeypots is complex and costly as they exclusively occupy an entire host operating system. The inability to detect attacks, so-called false negatives, is another issue encountered by these implementations. While the authors do not provide metrics around detection effectiveness, they do acknowledge that certain events are likely to cause false negatives, such as user interaction to trigger an attack or time bombs that delay an attack. Performance is another shortcoming of these implementations. Monitoring of state, which is employed by all these implementations, is an expensive operation. The performance of Honeymonkey in identification of exploits was about two minutes per URL, whereas the UW client honeypot averaged about 6.3 seconds per URL. Honeyclient has worse performance characteristics in the area of several minutes per URL.

Additional high interaction client honeypot projects are being begun [22, 11]. These projects acknowledge that web browsers are not the only clients that can be attacked. They also aim at using state-based knowledge to detect attacks, but plan to include a framework that is able to handle different types of clients (P2P, email, media players, etc).

3 Low Interaction Client Honeypots

The concept of low interaction client honeypot was first identified by us in our taxonomy of honeypots [17]. A low interaction client honeypot is a client honeypot that uses simulated clients instead of using a real system to interact with servers. The subsequent analysis of the response can be based on static analysis, such as signature matching and/or heuristics, which should lead to increased performance and the ability to detect some malicious responses that often elude traditional high interaction client honeypots, such as time bombs. Since the low interaction client honeypot makes use of a simulated client, containment of attacks is not a major concern and therefore simplifies deployment of the tool. However, at the same time, a low interaction client honeypot is likely to miss some unknown exploits that would be identified correctly by a high interaction client honeypot. The relationship of the low interaction client honeypot to the high interaction client honeypot is very similar to the interaction level of server honeypots. Tradeoffs exist, but the technologies are likely to complement each other.

We identified three tasks that a client honeypot has to fulfill. The client honeypot needs to interact with the server. This usually entails making a request to the server, consuming and processing the response. One portion of the client honeypot needs to create a queue of server requests for the tool to execute. One could employ several algorithms to create such a queue of server requests, such as crawling or search engine integration. In the end, the client honeypot needs to analyze the system or the server response for violation of the system's security policy after the client honeypot has interacted with the server. The first implementation of a low interaction client honeypot framework, called HoneyC [16], implements these requirements.

HoneyC is a platform independent framework and consists of three components that map to the functional requirements outlined above: Queuer, Visitor, and Analysis Engine as shown in Figure 3. Each of these components supports pluggable modules to suit specific needs. This is achieved by loosely coupling the components via a command redirection operator (pipe) for passing a serialized representation of the request and response objects. As long as the components agree on the serialized representation of the request and response object, this makes the components implementation independent and interchangeable. This technique is commonly used in Unix for a flexible approach to perform complex tasks. In our instance, this allows us to create a Queuer component that generates request objects via integration with a particular search engine API written in Ruby, or to implement a Queuer component that crawls a network in C.

In the initial version of HoneyC, we are concentrating on the HTTP 1.1 protocol [9] and have created a serialized representation of HTTP requests and HTTP responses for the components to interact with. We have chosen XML representation [7] as an industry-wide standard for exchanging content in a machine readable format. Figure 4 shows examples of HTTP requests that are generated by the Queuer component and consumed by the Visitor component. Figure 5 shows examples of the corresponding HTTP responses that are generated by the Visitor and analyzed by the Analysis Engine. Since we are working with a command redirection operator to pass the request and response from component to component, the component simply needs to

```
<httpRequests xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:noNamespaceSchemaLocation="HttpRequests_v1_0.xsd">
        <httpRequest>http://honeyc.sourceforge.net/</httpRequest>
        <httpRequest>http://developer.yahoo.com/search/index.html</httpRequest>
        <httpRequest>http://www.bleedingsnort.com/</httpRequest>
        <httpRequest>http://www.mcs.vuw.ac.nz/~cseifert/blog/index.php</httpRequest>
</httpRequests>
```

Figure 4: HTTP Requests Example

```
<httpResponses xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
   xsi:noNamespaceSchemaLocation="HttpResponses_v1_0.xsd">
        <httpResponse>
                <uri>http://honeyc.sourceforge.net/webBrowser/UnitTest.html</uri>
                <code>200</code>
                <body>&lt;!DOCTYPE HTML PUBLIC &quot;-//W3C//DTD HTML 4.01
                Transitional//EN&quot;\r\n&quot;http://www.w3.org/TR/html4/loose.dtd&quot
                ;&gt;\r\n&lt;html&gt;\r\n&lt;head&gt;\r\n&lt;title&gt;Untitled Document
                &lt;/title&gt;\r\n&lt;meta http-equiv=&quot;Content-Type&quot; content
                =&quot;text/html; charset=iso-8859-1&quot;&gt;\r\n&lt;/head&gt;\r\n\r\n
                &lt;body&gt;\r\nThis is a test page for the web browser unit test.rule2
                pcre\r\n&lt;/body&gt;\r\n&lt;/html&gt;\r\n</body>
        </httpResponse>
</httpResponses>
```

Figure 5: HTTP Responses Example

read from standard input and write to standard output. To ensure high performance, threads are used to simultaneously read from standard input and write to standard output. The HoneyC framework automatically links each component to ensure appropriate flow of information.

Implementation components that work with the HTTP objects are provided with the initial version of HoneyC. The Yahoo Search API component is an implementation of a Queuer. It is responsible for retrieving URLs by querying the Yahoo Search API with a set of given search parameters. The web browser is a simple implementation of a Visitor component that makes the HTTP request and passes on the response to the Analysis Engine. We implemented a Snort Rules Analysis Engine component, which is able to analyze the response against a set of Snort signatures, a de facto standard of intrusion detection signatures, originating from the lightweight intrusion detection system Snort [15].

4 Preliminary Results of HoneyC

In this section we present preliminary results of working with HoneyC. We examine performance figures first. With the initially provided HTTP components, we executed HoneyC to query the Yahoo Search API for 20 distinct keywords and

```
alert tcp any any <> any any (msg: "Sony CD First4Internet XCP uninstallation ActiveX
control identified on web page.";
reference:urlhttp://www.frsirt.com/english/advisories/2005/2454; sid:3400000; rev:1;
classtype:trojan-activity; content:" clsid:80E8743E-8AC5-46F1-96A0-59FA30740C51"; nocase;
flow:to_client,established; )
```

Figure 6: Snort Signature ActiveX Example

```
alert tcp any any <> any any (msg: "Web site contains code to modify your home page
(IE)."; reference:url,http://honeyc.sourceforge.net/signatureReferences.php; sid:3400001;
rev:1; classtype:trojan-activity; content:" javascript"; nocase; content:" setHomePage";
nocase; flow:to_client,established; )
```

Figure 7: Snort Signature JavaScript Example

subsequently had HoneyC visit 2000 websites. The responses of the websites were analyzed against a handful of Snort rules. The duration of HoneyC execution was averaged resulting in a visit and analysis of one website in 3.5 seconds, a performance improvement compared to traditional high interaction client honeypots. No direct comparison between a high interaction client honeypot and HoneyC has been undertaken at this time as experiments need to involve identical hardware and visitation schemes, and is left for future work.

We examined HoneyC's ability to detect malicious servers with the Snort Rule Analysis Engine. Figure 6 and 7 depict two examples of Snort signatures. Snort signatures are composed of matching elements and informational elements that are included in the alert that is being generated once a match has been determined. The first section *tcp any any ¡¿ any any* states that any tcp traffic should be considered. *flow:to_client,established* filters this traffic further into consideration of traffic that only flows to the client, aka the response of the server. Finally, the content specifies the specific string that should be contained in the response for the rule to fire. Optionally, content matching can be case insensitive. Once the rule fires, an alert is generated that contains the informational elements of the rule, such as the message, the unique identifier SID, the reference URL with additional information about the alert, and the classification of the alert.

The two signatures depicted support identification of malicious content in a web server response. The first signature identifies a malicious ActiveX control [8] using the class id as an identifier of the ActiveX component, whereas the second signature identifies EMACScript code [10], commonly referred to as JavaScript, that attempts to modify a user's home page setting of their browser. The latter is a good example of how HoneyC can identify malicious code that is difficult to identify by current high interaction client honeypots, since user interaction might be required to trigger the JavaScript code. How-

ever, depending on the signature quality, false alerts could result and this is a tradeoff to the higher performance obtained with low interaction client honeypots.

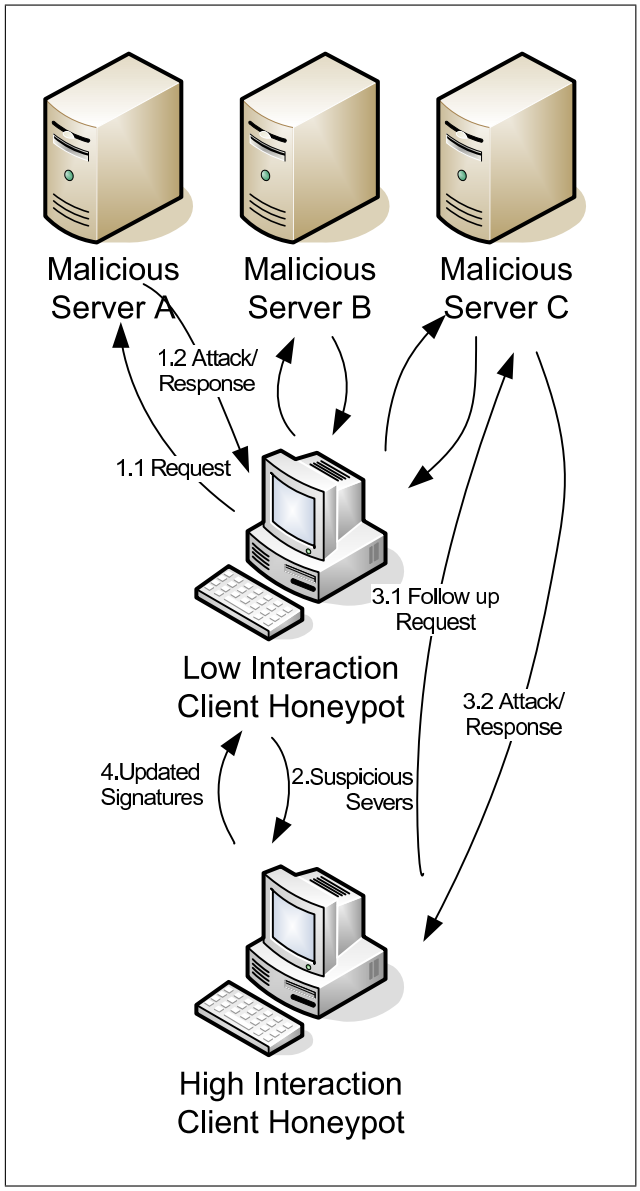

Figure 8: Complementary Client Honeypots

5 Conclusion and Future Work

In this paper we have introduced a new type of client honeypot, a low interaction client honeypot, which is designed to address some of the shortcomings of traditional high interaction client honeypots. We have provided and described an initial implementation of this new technology with HoneyC. While HoneyC is still in its initial stages, preliminary testing has shown some promising results in performance and detection capabilities of this new technology. We acknowledge that signature-based detection as currently implemented by HoneyC is likely to result in some false alerts. However, similar to server honeypots, we believe the performance capabilities of low interaction client honeypots will complement the detection rates of high interaction client honeypots when used in tandem.

Future work is targeted at investigating the complementary nature of low interaction and high interaction client honeypots. We envision a complementary system that makes use of both technologies to identify malicious servers fast and reliably as shown in figure 8. The low interaction client honeypot could serve as the primary system to crawl the network and identify malicious servers. Since the classification of this component could signify a false alert, the information about the suspicious server is passed to a high interaction client honeypot for a follow up interaction. Once the high interaction client honeypot has made a final assessment of the server, it could feedback this information to be combined with the initial assessment to tune the detection algorithm of the low interaction client honeypot. Further, the high interaction client honeypot could feed entirely new signatures to the low interaction client honeypot to be considered in its detection algorithm.

In addition, there are many opportunities in improving the existing implementation of HoneyC itself, such as performance improvements and creation of more comprehensive detection algorithms and signatures. Once implemented, they would allow us to perform some direct performance and detection capability comparisons between the two technologies. Further, there are additional opportunities for future work in the area of Queuer algorithms and Visitor components. We would like to investigate what Queuer algorithms are most effective in the selection of potentially malicious servers. Also, we would like to include different Visitors and compare distributions and trends of malicious web servers vs. other types of servers. If you are interested in joining the project and bringing HoneyC forward, please contact Christian Seifert at cseifert@mcs.vuw.ac.nz.

References

[1] Back Officer Friendly. Available from `http://www.nfr.com/resource/backOfficer.php`; accessed on 6 June 2006.

[2] NetFacade. Available from `http://www22.verizon.com/fns/solutions/netsec/`; accessed on 2 June 2006.

[3] CyberCop Sting, 1999.

[4] Honeywall CDROM Eyeore, 2003. Available from `http://project.honeynet.org/tools/cdrom/eeyore/download.html`; accessed 6 July 2006.

[5] Cheswick, B. An Evening with Berferd in which a cracker is Lured, Endured, and Studied. In *Winter 1992 USENIX Conference* (San Francisco, 1992), USENIX, pp. 163–174.

[6] Cohen, F. Deception Toolkit. Available from `http://all.net/dtk/dtk.html`; accessed on 6 July 2006.

[7] Consortium, W. W. W. W. Extensible Markup Language (XML), 1998. Available from `http://www.w3.org/XML/`; accessed on 12 August 2006.

[8] Corp, M. ActiveX Controls, 1996. Available from `http://msdn.microsoft.com/library/default.asp?url=/workshop/components/activex/activex_node_entry.asp`, accessed on 10 August 2006.

[9] Fielding, R., Gettys, J., Mogul, J. C., Frystyk, H., Leach, P., and Berners-Lee, T. Hypertext Transfer Protocol – HTTP/1.1, 1999. Available from `http://tools.ietf.org/html/rfc2616`; accessed on 10 August 2006.

[10] International, E. C. M. A. Standard ECMA-262 - ECMAScript Language Specification , 1999. Available from `http://www.ecma-international.org/publications/standards/Ecma-262.htm`; accessed on 12 August 2006.

[11] Mara, F., Tang, Y., Steenson, R., and Seifert, C. Capture - Honeypot Client, 2006. Available from `http://capture-hpc.sourceforge.net/`; accessed on 12 August 2006.

[12] Moshchuk, A., Bragin, T., Gribble, S. D., and Levy, H. M. A Crawler-based

Study of Spyware on the Web. In *13th Annual Network and Distributed System Security Symposium* (San Diego, 2006), The Internet Society.

[13] Pouget, F., and Holz, T. A Pointillist Approach for Comparing Honeypots, 2005.

[14] Provos, N. Honeyd Virtual Honeypot. Available from `http://www.honeyd.org/`; accessed on 6 July 2006.

[15] Roesch, M. Snort-Lightweight Intrusion Detection for Networks . In *13th Large Systems Administration Conference* (Seattle, 1999), Usenix, pp. 229–238.

[16] Seifert, C. HoneyC - The Low-Interaction Client Honeypot, 2006. Available from `http://honeyc.sourceforge.net`; accessed on 12 August 2006.

[17] Seifert, C., Welch, I., and Komisarczuk, P. Taxonomy of Honeypots, July 2006 2006. Available from `http://www.mcs.vuw.ac.nz/comp/Publications/index-byyear-06.html`; accessed on 14 July 2006.

[18] Spitzner, L. *Honeypots: Tracking Hackers*. Addison-Wesley, Boston, 2002.

[19] Stoll, C. Stalking the Wily Hacker. *Communications of the ACM 31*, 5 (1988), 484–497.

[20] Wang, K. Honeyclient, Version 0.1.1. Available from `http://www.honeyclient.org/`; accessed on 6 July 2006.

[21] Wang, Y.-M., Beck, D., Jiang, X., Roussev, R., Verbowski, C., Chen, S., and King, S. Automated Web Patrol with Strider HoneyMonkeys: Finding Web Sites That Exploit Browser Vulnerabilities . In *13th Annual Network and Distributed System Security Symposium* (San Diego, 2006), Internet Society.

[22] Yuan, B., and Holz, T. Client-Side Honeypots, 2006. Available from `http://pi1.informatik.uni-mannheim.de/diplomas/show/27`; accessed on 12 August 2006.