APHIDS: A Mobile Agent-Based Programmable Hybrid Intrusion Detection System

Ken Deeter[1], Kapil Singh[1], Steve Wilson[1], Luca Filipozzi[2], and Son Vuong[1]

[1] Department of Computer Science
[2] Department of Electrical Engineering
University of British Columbia
Vancouver, Canada

Abstract. Intrusion detection systems are quickly becoming a standard requirement in building a network security infrastructure. Although many established techniques and commercial products exist, their effectiveness leaves room for improvement. We propose an intrusion detection system architecture which takes advantage of the mobile agent paradigm to implement a system capable of efficient and flexible distribution of analysis and monitoring tasks, as well as integration of existing detection techniques. Our architecture defines a high-level application specific scripting language to specify the interaction between monitoring agents and analysis agents.

1 Introduction

Intrusion detection systems (IDS) are quickly becoming a standard component of network security architectures, complementing conventional techniques such as firewalls, encryption, and authentication. The purpose of an IDS is to detect and report unusual and malicious behavior in a network – often caused by the intentional circumvention of an existing security measure by a malicious user.

This task requires a system to be able to characterize patterns in the context of a complex network environment. As networks gain in size, complexity, and variation, the task of analyzing different parts of the network and maintaining a cohesive view of the entire system becomes increasingly difficult. While the initial problems of performance have been overcome, the effectiveness of current intrusion detection systems remains limited. The primary difficulties include the management and correlation large amounts of data and the frequent occurrences of *false positives* – when a system identifies harmless behavior as malicious.

This paper pressents a novel IDS architecture, which we call APHIDS, that employs mobile agents to perform monitoring and analysis in a distributed and timely manner. This architecture delegates data capture and detection tasks to existing monitoring systems. Distributed search and analysis tasks are implemented with mobile agents, and the system provides a high level scripting facility to define how analysis results from these agents should be combined and reported. Our approach is complementary to existing IDS techniques – it aims to effectively combine the strengths of existing technologies to provide more effective results.

This paper is structured as follows: the following section provides a background for readers not familiar with intrusion detection and the mobile agent paradigm. Section 3 summarizes our motivations and design goals for the system. Section 4 details our system architecture and section 4.2 introduces our automation system known as *distributed correlation scripts*. Finally, several conclusions and ideas for future work are offered.

2 IDS Background and Related Work

Intrusion detection techniques are largely classified into two areas: *misuse* detection and *anomaly* detection. In the *misuse* approach, malicious behavior is characterized and expressed in machine readable form, and the IDS checks for the existence of this behavior using a deterministic method. The work of specifying the malicious behavior is left entirely to the developers and users of the system. An example of such a system is the Snort network IDS [1]. Snort is configured using a database of *signatures* which characterize network packets that are potentially malicious. Using this database, Snort monitors a network connection and logs all occurrences of network packets that match any of the configured signatures. As is the case with any *misuse* based system, however, Snort cannot detect events for which no signatures have been developed.

Anomaly detection refers to an approach where a system is trained to learn the "normal behavior" of a network. An alarm is raised when the network is observed to deviate from this learned definition of normality. This type of system is theoretically capable of detecting unknown attacks, overcoming a clear limitation of the *misuse* approach. However, because an alarm is based on a detected change in an abstract representation of the network behavior, information about the root cause of the deviation may be difficult to infer.

Intrusion detection systems are also traditionally classified as either *network-based* or *host-based*. *Network-based* systems monitor network traffic and inspect packet transmissions for suspicious behavior. A network-based system can be used to provide detection for multiple hosts by locating the monitoring component appropriately (at a network ingress point, for example). *Host-based* systems operate on single hosts, and operate on low-level system data, such as patterns of system calls, file access, or process usage. They can monitor for suspicious behavior, or they can scan configurations to detect potential vulnerabilities.

Log correlation refers to the process by which an IDS combines captured data that is distributed both spatially and temporally, and tries to extract significant and broad patterns. For example, a similar type of attack detected at different points in time may indicate an automated, coordinated attack. Likewise, an event detected at different monitoring locations may be indicative of a distributed attack. In general, the more data that can be collected related to a specific event, the easier it is for a security administrator to respond in an effective manner. The automatic correlation process reduces the need for an administrator to search through large log files manually, saving valuable time.

The established approach to the log correlation process involves the collection of distributed sensor data into a central location, and the application of searching and data aggregation techniques to discover patterns.

2.1 Agent-Based IDS Research

The applicability of mobile agents and software agents to intrusion detection has been explored in several other projects. The AAFID [2] [3] system from Purdue University's COAST Laboratory (now CERIAS) introduced an autonomous agent-based IDS, which formed a reference for comparison for many of the mobile agent-based systems introduced since. Related work on this system has been reported recently as well [4].

The key differentiating factor between the AAFID approach and the mobile agent-based approach is the mobility of the agents participating in the IDS. Many independent projects have explored the advantages of the mobility aspect of mobile agents in the context of intrusion detection. The IDA system [5] employs mobility to identify the source of attacks. Systems which use mobile agents to model biological immune systems have been proposed [6] and elaborated on[7]. The Micael system [8] and the MA-IDS system [9] both use mobile agents to aggregate distributed information related to attacks. The Sparta system [10] uses mobile agents to provide a query like functionality to reconstruct patterns of related events distributed across multiple hosts. The MAIDS project at Iowa State University explored the usage of dynamic agent composition techniques to create an array of lightweight agents to perform a full range of IDS-related tasks.

The APHIDS architecture presented in this paper represents a variation of the existing mobile agent-based approaches (with some similarities to the SPARTA and MAIDS systems). Our system shares a common goal of exploiting the mobility of the agents to perform distributed correlation. It differs, however, in the mechanism by which these mobile agents are coordinated and combined. APHIDS provides a scripting capability that aims to automate evidence gathering tasks that system administrators would otherwise perform manually. Our system also attempts to utilize and integrate existing IDS technologies and implementations instead of replacing their functionality.

3 Motivation

The initial motivation for our work is to address limitations of current IDS systems by taking advantage of the mobile agent paradigm. Specifically, we address the following limitations of conventional centralized approaches to the log correlation task:

– *Bandwidth Scalability*: The bandwidth required to collect large, distributed data sets from distributed sensors can pose a significant overhead cost, affecting network performance.

– *Processing Scalability*: The processing capability of the centralized approach is limited by the computational power of a single analysis center, even though other resources may be available.
– *Analysis Delay*: In the centralized approach, logs are collected only periodically, delaying results by at most one collection and analysis cycle. Long delays can hamper a timely and effective response.
– *Integration*: Existing commercial IDS's are sold and developed as standalone products, and they do not support aggregation of data between various systems. [3]

4 APHIDS Framework

4.1 Network-level Architecture

From a deployment perspective, the structure of our system is trivial. It requires the placement of an agent engine at every relevant location, including network monitoring devices, web servers and other service-providing hosts, as well as hosts running other IDS systems[4]. APHIDS is realized as a distributed layer which operates on top of a set of distributed agent engines. This design allows us to access network components in a uniform and efficient manner, and allows all of our system components to take advantage of the benefits provided by the agent platform. Mobile agents are used for monitoring the output from other IDS systems, for querying the log files and system state, and for reporting results. These various agents are coordinated using a *distributed correlation script*, which is described in section 4.2.

4.2 Distributed Correlation Scripts

A key feature of our system is that it allows the specification of coordinated analysis tasks using a high-level specification language. This language is used to define a *Distributed Correlation Script* (DCS), which associates a *trigger event* with a series of *analysis tasks* to be performed when the event is detected. A collection of these scripts is provided as input to the system.

4.3 Detection and Analysis

Our system's detection and analysis procedure for a particular attack can be broken down into the three following steps:

1. Detection of a *trigger event*[5]

[3] Competing commercial IDS's have no incentive to cooperate with each other, even though the data they each collect may yield better results when combined.
[4] Our architecture also requires the existence of one a system console, used for system configuration and display of analysis results
[5] A *trigger event* is an abstract concept that simply refers to any suspicious event occurring on the network. The exact definition of a *trigger event* is left up to a *trigger agent* that is programmed to detect it.

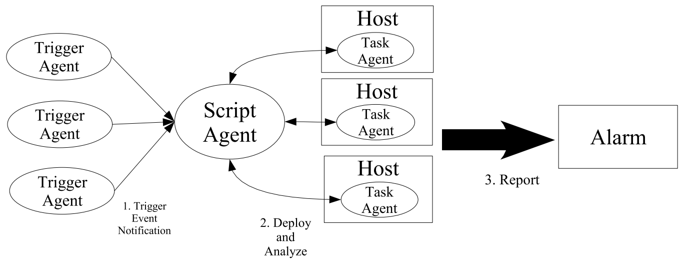

Fig. 1. This diagram shows the interactions between agents in our system. After deployment, trigger agents notify the script agent of occurrences of trigger events. The script agent then launches task agents which migrate to appropriate locations to perform analysis, the results of which are returned to the script agent. The script agent finally conditionally raises an alarm and generates a report

2. Collection and Analysis of data related to this event
3. Reporting the event

Each DCS specifies exactly one *trigger agent* (along with some initialization parameters) and the types of analysis that should be performed based on a trigger notification from that agent. The tasks are specified by referring to mobile agents called *task agents* that implement each type of analysis task.

When a particular script is enabled, an agent is created to manage it. This *script agent* is responsible for instantiating the *trigger agent*, launching the task agents, and storing the data returned by these various agents. When a trigger agent detects that a trigger event has occurred, it sends a set of values describing the event back to the script agent to which it is associated. The script agent in turn launches task agents according to the script specification. The values received from the trigger agent are made available to these analysis agents. Data returned from each task agent is stored by the script agent and is made available to subsequently launched task agents. The manner in which these values are passed to task agents is determined in each script, allowing for different forms of collaboration to be specified by writing different scripts.

After all the task agents have completed, the script agent can make a decision to report an alarm. This decision process is also specified in the script (the details are described in the following subsection). This entire process is illustrated in fig. 1.

These scripts allow users and developers to freely mix and match existing monitoring and analysis agents. The correlation task represented by each script is capable of considering data and conditions from distributed sources.

4.4 Script Syntax

The syntax of a distributed correlation script is shown and described below:

```
Trigger <TriggerAgent>(<Parameters>...) : <RetValueName> {
    Task <DataTaskAgent>(<Parameters> ...) : <RetValueName>
      :
}
alarm {
  <ConditionTaskAgent>(<Parameters> ...) : <ConditionValueName>
    :
  condition((<ConditionValueName> AND <ConditionValueName>)
      OR (NOT <ConditionValueName>))
}
```

A script is composed of a combination of a `Trigger` block and `alarm` block. The `Trigger` block specifies a trigger agent name, along with initialization parameters. The trigger block contains a list of task agents that are to be launched when a trigger event notification is recieved. Each task is specified with the `Task` keyword, the name of the agent which performs the task, and configuration parameters for that agent. We refer to the agents within this block as *data collection agents*.

To facilitate the sharing of results between agents (including information related to the trigger event returned by the trigger agent), each agent is allowed to return an associative array of values, which is referenced by the identifier provided after the colon following the agent's invocation (notated by `<RetValueName>`).

In the `alarm` block, a second set of task agents are specified. These agents however, can only return a boolean value, and are used only to check for the existence of certain conditions in the network. These agents are referred to as *condition checking agents*. The return value of these condition checking agents are also named and referenced by a variable name provided at the end of the each agent's invocation.

Finally, the alarm block concludes with a `condition` statement. The statement is composed of an arbitrary boolean combination of condition values defined in the same alarm block. This condition determines whether an alarm is generated for this event.

A design decision was made to ensure that none of the analysis or monitoring implementations are directly specified in a DCS. Instead, a script can only refer to an agent that implements the required functionality. This choice allows complex monitoring and analysis tasks to be implemented in a more suitable language (e.g. Java). We wished to avoid replicating an entire programming language to allow complex computational operations to be specified in the scripts themselves. This design also allows the system to be extended in a straightforward manner. For example, new trigger and task agents can be defined after the APHIDS is initially deployed, and new scripts referring to those agents can be incorporated on the fly, without requiring any changes to the existing set of scripts and agents.

4.5 Analysis Scenarios

To illustrate the practical usage of a DCS we present two analysis scenarios that can be handled by a DCS.

In the first example we consider the following scenario: an attacker from host A has gained access to a SSH account on one of several protected servers. In an attempt to infiltrate other hosts, the attacker performs a portscan (also from host A) of the network. This scenario would be detected by APHIDS by deploying the following script:

```
Trigger SnortPortScan() : PS {
    Task ArgusFindConnections(PS.source, PS.timeRange) : FC
}
alarm {
    LoginCheck(PS.source, FC.targets, PS.timeRange) : C1
    condition(C1)
}
```

This script would first deploy a trigger agent written to capture notifications of a port scan from a Snort system. Once the port scan is detected the trigger agent would notify the script agent. The script agent would then deploy a data collection agent to perform the `ArgusFindConnections()` task. The `PS.source` argument is passed in from the trigger agent to tell the data collector where the port scan originated, as well as the `PS.timeRange` argument to tell the agent what time range to look for connections (we would be interested in a time before the port scan occurred). The data collection agent would then migrate to a host running the Argus network monitor, and discover that a number of SSH connections have been made from the host A to a local host (or possibly several hosts). Based on this information, a condition checking agent implementing the `LoginCheck()` task is deployed. This agent uses the parameter FC.targets, to sequentially move to the hosts that host A has connected to, and checks for login attempts from PS.source. In our example the login from the attacker would be found which would raise our alarm. As a result, the system administrator would be alerted to the fact that a certain user launched a port scan and accessed the network successfully, indicated a potential security breach. The administrator could then investigate further and may discover a compromised user account.

For our second example we consider the following scenario: an attacker makes an unusually high number of connections to a server on the monitored network, possibly with the intent to perform a denial of service attack against a host, or inidicating that the attacking host has been infected with a virus or worm. The following script would respond to this attack:

```
Trigger ConnectionOverLoad(): CL {
    Task CheckUserNames(CL.target, CL.source) : CN
}
alarm {
    UserCount(CN.nameList) : C1
    condition(C1)
}
```

This script would deploy a trigger agent that implements an anomaly-type detection method or interfaces with an external detection system. Once the rate of connections from a certain host reached a defined threshold the trigger agent would report back to the script agent. In response, the `CheckUserNames()` data collection agent would be invoked to collect the list of user names that have logged into `CL.target` from `CL.source`. Finally, the `UserCount()` agent would be deployed to confirm whether a particular user has performed an extraordinarily large number of logins. If a suspect is found, an alarm will be raised. The system administrator could respond by enabling process accounting and auditing for the user in question.

As can be seen from our examples, the actual analysis tasks in our system are always performed by mobile agents that have access to system libraries and that can be written in an expressive programming language. The DCS mechanism only provides a mechanism to coordinate these agents, and combine their results in a simple manner useful for intrusion detection.

5 Discussion

First, we address the specific limitations described by the central approach.

– *Bandwidth Scalability*: APHIDS uses a remote evaluation technique based on mobile agents to remove the need to transfer log data from sensors to an analysis center. If analysis agents are sufficiently small (typically less than 20 kilobytes) when compared to the log data size (often several gigabytes), then bandwidth savings can be realized.
– *Performance Scalability*: Because mobile agents perform distributed search and analysis, the processing work related to each incident is inherently distributed, reducing the amount of work done per host and removing a single host bottleneck.
– *Analysis Delay*: Because each incident is analyzed immediately, the upper bound on the delay between an incident occurence and its corresponding report depends only on the processing time needed for analysis of one incident, as well as the time needed to transfer and collect related data. Delay can be further reduced by launching data collection agents in parallel where possible.
– *Integration*: Our architecture's use of an agent engine guarantees that new components (new IDS's, monitoring systems, etc.) can be integrated without requiring global changes. Our system provides a uniform layer to analysis agents allowing them to effectively combine data from different sensors.

Next we identify other benefits that are realized by our framework:

– Because of the distributed and integrative nature of our framework, it can be adapted to an existing security infrastructure.

– Distributed correlation scripts are able to capture the expert knowledge of security administrator by automating the standard investigative procedures that are performed in response to an incident.
– APHIDS is able to discover and utilize greater amounts contextual information when processing an incident, allowing it to potentially make more informed decisions.
– Our framework is extensible. For example, a user can introduce new types of trigger, script, and data collection agents, as well as provide new scripts to combine existing agents in novel manners.

The mobile agent approach has several disadvantages which must also be kept in mind:

– Virtualization and serialization routines required for agent mobility cause performance overhead that may be significant without proper design consideration.
– Security of the agent system must be considered, including the potential to perform a denial-of-service attack by intentionally triggering analysis agents.

6 Conclusions and Future Work

We have presented a mobile agent-based architecture for a programmable IDS which incorporates existing IDS technologies. Our next direct step will be to complete our prototype implementation of the DCS system.

To our knowledge, our IDS architecture is the first mobile agent-based architecture with a explicit primary goal to use mobile agents to form a middle-ware which ties together separate network and host-based intrusion detection systems. By using a mobile agent platform that allows us to incorporate object-oriented design techniques, we are able to treat the agent engines as an uniform interface for agents to access data generated by each type of monitoring system.

Our system architecture realizes the scalability of mobile agent-based approaches, and addresses flexibility, extensibility, and delay limitations of existing approaches. Our two examples presented in section 4.5 illustrate how APHIDS can incorporate both *misuse* and *anomaly* approaches by implementing agents that employ each technique.

As our current work focuses on the distributed correlation implementation, in the future we would like to investigate enhancement of the detection portion of the framework. Detection mechanisms are modularized by the trigger agent, allowing for the investigation of new detection techniques. This abstraction also allows the framework to incorporate advanced finite state machine-based detection systems such as the Bro system [11] and STAT-based systems [12]. We intend for APHIDS to become a useful framework for experimenting with new intrusion detection techniques, while making the advantages of the mobile agent paradigm available to new implementations.

7 Acknowledgments

This research is supported in part by the Directorate of Telecom Engineering and Certification of the Department of Industry Canada. We would like to thank Peter Chau, Os Monkewich and Sergio Gonzalez Valenzuela for fruitful discussions and for reviewing the drafts of the paper.

References

1. Roesch, M.: Snort – lightweight intrusion detection system for networks. In: Proceedings of USENIX LISA'99. (1999)
2. Crosbie, M., Spafford, G.: Defending a computer system using autonomous agents. In: 8th National Information Systems Security Conference. (1996)
3. J S Balasubramaniyan, J.O.G.F., Isacoff, D., Spafford, E., Zamboni, D.: An architecture for intrusion detection using autonomous agents. Technical Report 98/05, COAST Laboratory, Purdue University (1998)
4. Wu, Y.S., Foo, B., Mei, Y., Bagchi, S.: Collaborative intrusion detection system (cids): A framework for accurate and efficient ids. In: Proceedings of the 19th Annual Computer Security Applications Conference (ACSAC'03). (2003)
5. Asaka, M., Taguchi, A., Goto, S.: The implementation of ida: An intrusion detection agent system. In: Proceedings of the 11th FIRST Conference. (1999)
6. Faukia, N., Billard, D., Harms, J.: Computer system immunity using mobile agents. In: HP Openview University Association 8th Annual Workshop. (2001)
7. Faukia, N., Hassas, S., Fenet, S., Albequerque, P.: Combining immune system and social insect metaphors: A paradimg for intrusion detection and response system. In: Proceedings of the 5th International Workshop for Mobile Agents for Telecommunication Applications. (2003)
8. Duarte de Queiroz, J., Fernando Rust da Costa Carmo, L., Pirmez, L.: Micael: An autonomous mobile agent system to protect new generation networked applications. In: 2nd Annual Workshop on Recent Advances in Intrusion Detection. (1999)
9. Li, C., Song, Q., Zhang, C.: Ma-ids architecture for distributed intrusion detection using mobile agents. In: Proceedings of the 2nd International Conference on Information Technology for Application (ICITA 2004). (2004)
10. Kruegel, C., Toth, T.: Sparta – a mobile agent based intrusion detection system. In: Proceedings of the IFIP Conference on Network Security (I-NetSec). (2001)
11. Paxson, V.: Bro: A system for detecting network intruders in real-time. Computer Networks **31** (1999) 2435–2463
12. Vigna, G., Kemmerer, R.A.: Netstat: A network-based intrusion detection system. Journal of Computer Security **7** (1999)