APHIDS: A Mobile Agent-based Programmable Hybrid Intrusion Detection and Analysis System

by

Ken Deeter
B.S.E., Princeton University, 2002

A THESIS SUBMITTED IN PARTIAL FULFILMENT OF THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF SCIENCE

in

THE FACULTY OF GRADUATE STUDIES
(Department of Computer Science)

We accept this thesis as conforming to the required standard

THE UNIVERSITY OF BRITISH COLUMBIA
September 2004

Library Authorization

In presenting this thesis in partial fulfillment of the requirements for an advanced degree at the University of British Columbia, I agree that the Library shall make it freely available for reference and study. I further agree that permission for extensive copying of this thesis for scholarly purposes may be granted by the head of my department or by his or her representatives. It is understood that copying or publication of this thesis for financial gain shall not be allowed without my written permission.

Ken Deeter — Name of Author *(please print)*

24/09/2004 — Date (dd/mm/yyyy)

Title of Thesis: APHIDS: A Mobile Agent-based Programmable Hybrid Intrusion Detection and Analysis System

Degree: Master of Science Year: 2004

Department of Computer Science
The University of British Columbia
Vancouver, BC 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. This thesis documents a design and prototype implementation of a modular, mobile agent-based intrusion detection framework, known as APHIDS. This framework provides a platform for performing distributed monitoring, search, and analysis tasks while realizing the benefits of the mobile agent model. Its modular design allows simple extension and adaptation to a large variety of scenarios. Several baseline performance measurements are accompanied by a theoretical framework to describe the performance implications of using mobile agents for intrusion detection-related tasks.

Contents

List of Figures

List of Tables

Acknowledgements

I thank Dr. Son Vuong for his helpful supervisorship, Dr. Charles Krasic for his helpful revisions and feedback, Sergio Gonzalez-Valenzuela, Christian Chita, Shahed Alam, Steve Wilson, Kapil Singh, and Luca Filipozzi, for their ideas and collaboration, and Qian Li for her moral support during the research process.

This research is supported in part by the Directorate of Telecom Engineering and Certification of the Department of Industry Canada. I would like to thank Peter Chau and Os Monkewich and for fruitful discussions and feedback regarding this work.

Ken Deeter
September, 2004

Chapter 1

Introduction

Intrusion detection systems are quickly becoming a standard component of network security architectures, complementing conventional techniques such as firewalls, encryption, and authentication. This thesis describes a novel architecture for building an intrusion detection system: one that employs mobile software agents to coordinate monitoring, data collection, and analysis tasks.

1.1 Motivation

A useful starting point would be to explore and define the term *intrusion.* In the physical world, the word implies a purposeful, unwanted entry – a break-in. This carries a negative connotation, but it is important to note that the situations in which an entry becomes an intrusion are entirely subjective.

For example, a friend entering a house to water the plants while the owners are on vacation would not be thought of as an intrusion (although an uninformed neighbor may feel otherwise). However, if the same friend enters the bedroom and begins to sort through personal belongings, it could be considered as one. As this example aims to illustrate, when we begin to identify real-world intrusions, we must keep in mind the context in which we define intrusions. Answers to questions such as "What is being intruded into?" or "What is being protected?" must be clearly answered and defined

before we can identify particular actions as intrusions.

Intrusion Detection, as the name implies, is the process of detecting an intrusion. An Intrusion Detection System, likewise, is a system intended to perform or help perform Intrusion Detection. Many real-world organizations have physical intrusion detection systems. Large corporate offices often have an array of video surveillance cameras monitored by security officers and managers. Even the secretary at the front desk can be thought of as the first "monitor" in the physical intrusion detection process. Many homes now have home security alarms installed, which when enabled, monitor for unwanted intruders. The combination of a network of security personnel, security procedures, and monitoring systems make up what can be thought of as a real-world Intrusion Detection System.

In the context of computer systems, an IDS refers to a system that is required to continually monitor computer systems or networks and identify patterns of events as intrusions while considering the context of the operating environment. Protection of a system is especially significant when it is exposed to the public Internet and is accessible from any host world-wide. Although technologies such as firewalls, encryption, and authentication exist, experience has shown that an extra layer of protection is often necessary. The aforementioned technologies attempt to limit access to a given resource, but do not consider the possibility of unwanted use of a resource, once access has been gained. Here, the term "unwanted" has several definitions. Below are some possible origins of such unwanted usage behavior:

- Malicious Users: Internet attackers have many reasons to attempt to gain unauthorized access to a system, ranging from personal financial gain through data theft or theft of service to simple bragging rights.

- Denial of Service: malicious users or organizations may wish to intentionally render a resource unavailable, typically by rapidly repeating a large number of *valid* requests.

- Viruses and Worms: the undesirable activity may not be cased by a user, but a malicious software program. Often these programs use

unauthorized access to resources to further replicated and spread themselves.

- Mis-configuration: Although a network may employ other security technologies, misconfigurations in the installations may leave exploitable holes, allowing for unintended usage.

The environments that need to be protected exist in a large range of sizes and architectures. Thus, finding generic techniques that are applicable to a majority of the situations is difficult.

The understanding of "context" by an IDS is just as important as the example of the helpful friend watering plants described above. Just as the actions of the friend must be considered in the context of which he/she is acting, the events captured by and IDS must be analyzed while taking into account the specific characteristics and purpose of the monitored system. The greatest difficulty in creating an effective intrusion detection system lies in the problem of dealing with this "context", as the number of different contexts is virtually infinite. When the context is characterized incorrectly, the system can make two kinds of mistakes: a *false positive* in which the system reports a normal incident as an intrusion, and a *false negative* in which the system does not report a real intrusion.

In small-scale situations, it is still the current situation that humans are more effective in making judgements. However the immense amount of data generated by increasingly larger networked systems has rendered the reliance on human judgement impractical. The goal of the IDS, therefore, is to aid a human in monitoring and protecting a networked system.

It is important to note, however, that the role of the human operator is likely to always remain an essential part of the system. This will remain true even if only for the reason that, in the end, the human must define what is to be protected, which behavior is desirable, and likewise what is undesirable. From the perspective of the IDS (in the absence of a humanly intelligent IDS), the human operator always defines the context, which in turn determines how and what the system monitors and reports events.

An effective IDS, then, can be thought of as a system that does well

in helping the human user interested in monitoring for unwanted behavior. This can be broken down into several requirements:

- There must be an effective mechanism to communicate, from the human to the system, the context in which the system operates
- The system should be able to perform repetitive tasks and data collection automatically, to allow the user to concentrate on higher level problems
- The system should be able to capture an 'expertise' in network security analysis processes
- When a system is unable to determine corrective action automatically, the system should strive try to communicate to the user *all* the relevant data pertaining to an incident, and *none* of the irrelevant, to allow the user to implement appropriate changes.

This thesis documents an IDS architecture that attempts to meet the above requirements. It combines several existing IDS techniques with a mobile agent-based distributed analysis framework to integrate and extend the analysis capabilities of existing systems.

1.2 Thesis Contributions

This thesis provides five main contributions[1]:

- An exploration of the applicability of mobile-agent based design techniques to the design of an intrusion detection system. A description of a system design which utilizes the unique features of the mobile agent programming paradigm to overcome limitations of existing IDS designs.
- A prototype implementation of a such a system, and an account of the low-level technical decisions required to complete such a system.

[1] Components of this work have also been recently accepted for publication. [8, 7]

- A mathematical framework to characterize situations in which the mobile agent-based design provides superior performance characteristics over the existing designs.
- Empirical analysis results of the developed prototype.
- A a visualization tool created to analyze the complex behavior of mobile agent systems.

1.3 Thesis Structure

The thesis, excluding this introductory chapter and a concluding chapter, is divided into three main chapters. The second chapter provides a comprehensive background of intrusion detection systems and techniques, mobile agent technology, and related work concerning the application of the latter to the former. The third chapter documents the APHIDS architecture and details a prototype implementation. The fourth chapter provides theoretical and empirical analysis, as well as qualitative analysis describing the advantages of the APHIDS design.

Chapter 2

Background

2.1 A Taxonomy of Intrusion Detection Systems

The complexity of the intrusion detection problem has given rise to a variety of different approaches. Over time, these approaches have been classified and abstracted. Reviewed in this section are the two common classifications for existing intrusion detection systems.

2.1.1 Misuse v.s. Anomaly

One such classification concerns the techniques by which systems detect intrusions. The classification defines two categories: *misuse* detection and *anomaly* detection.

Misuse detection refers to an approach where malicious behavior is characterized and expressed explicitly in some machine readable form, and the system checks for the existence of this behavior using a deterministic method. The work of specifying and expressing the malicious behavior is left entirely to the developers and users of the system.

Currently, the most common intrusion detection method utilizes a misuse type method, over a network-based architecture (see Section 2.1.2). In this combination, the IDS uses a set of expressions known as *signatures* which serve to specify patterns of suspicious network transfers. When a transfer matching a known pattern is observed, an alarm is raised. Abstractly, these

signatures can be thought of as regular expressions for network traffic. The implementation of such a system, however, can be non-trivial due to the many different protocols and traffic patterns that need to be monitored.

An example of such a system is the Snort network IDS [34, 36]. Snort is configured using a public database of attack signatures. Using this database, Snort monitors a network connection and logs all occurrences of network packets that match any of the enabled signatures. If so configured, Snort can detect all attack instances that are described in the public signature database. However, as is the case with any *misuse* based system, Snort cannot detect events for which no signatures have been developed. Also, Snort does not attempt to correlate detected events with data from other monitoring systems.

Another similarly network-based technique is known as *vulnerability scanning*. Often a system administrator may misconfigure a system or groups of systems, leading to a unwanted services being offered to other hosts that become potential entry points for an attacker. A vulnerability scanner can automatically probe a network and identify services running on every host, so that this data can be compared to an intended configuration. These types of scanners can also check for unwanted services that have been started by non-privileged users – a typical symptom of a unauthorized access. The scanners, however, fall in the *misuse* category, and need to be provided with a database of checking procedures.

A third misuse type approach is known as *policy driven* detection. This approach defines policies to explicitly limit resource usage and access by software processes and users. When a policy is violated, an alarm is raised. This technique has the advantage of allowing operators to tailor policies towards specific applications, and the system is able to detect unknown attacks if they result in a policy violation. An example of such a system is the Blue-box system, developed at IBM[4].

Anomaly detection refers to an approach where a system is designed to identify behavioral anomalies within a network. Typically, such as system is trained to learn the "normal" behavior of a system. This training can focus on various aspects of the system, including system calls, users, process be-

havior, or network traffic. When the system is observed to deviate from this learned definition of normal, an alarm is raised. Various techniques borrowed from the machine learning and artificial intelligence (statistical modeling, feature extraction, genetic programming) fields have been considered in this approach [14, 5].

This type of technique is theoretically capable of detecting unknown attacks (both entirely novel attacks, and those for which no *misuse* type detection method has been developed), overcoming a clear limitation of the *misuse* approach. However, because an alarm is based on a detected deviation in some abstract representation of the actual system, the root cause of an alarm may be difficult to identify.

2.1.2 Network v.s. Host-based

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, on the other hand, operate on single hosts, and utilize data available at locally at the system. This requires that the IDS be installed directly on to each protected computer, and that it is allowed to share resources with other services running on each host. Despite the increased resource consumption, however, a host-based system may often have access to system data that is either unavailable or impractical to retrieve over the network, thereby potentially offering more effective or relevant results.

2.2 IDS Deployment Scenarios

To understand the differences between IDS designs, it is useful to keep in mind common real-world deployment scenarios and configurations. This section explores two such configurations: the *single-site, multi-backbone* (SSMB) scenario, and the *multi-site* (MS) scenario.

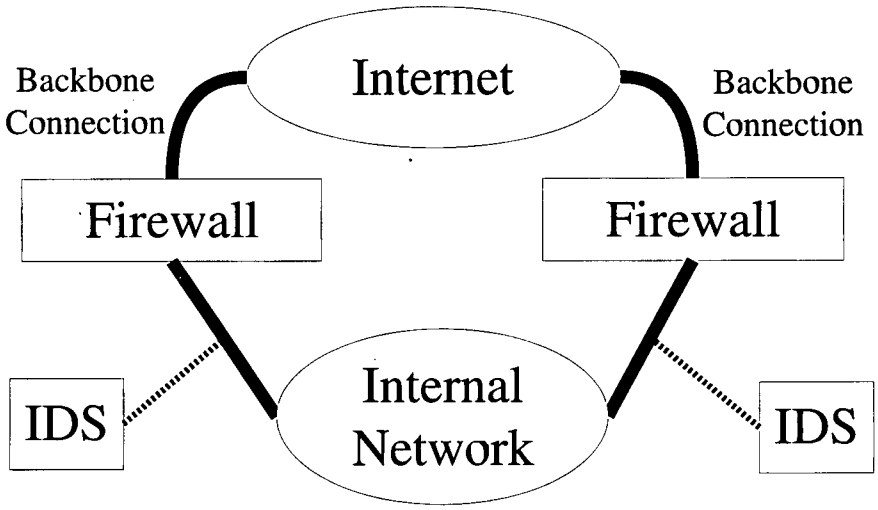

Figure 2.1: An example topology diagram of a *single-site, multi-backbone* scenario, in which an internal network is connected to the internet through two backbone links.

The SSMB scenario (shown in Figure 2.1) approximates a large enterprise network configuration. An internal network is protected by firewalls at ingress points, and the network has multiple backbone connections to the internet to provide redundancy in the case of backbone failure. An network-based IDS deployment in such a topology is typically done just "behind" the ingress firewalls. This location is chosen for several reasons:

- The firewall can block out a majority of irrelevant and unwanted traffic. As an IDS can typically handle only a fraction of the bandwidth that a firewall is able to, allowing the IDS to process only the filtered traffic results in a more scalable design.
- To protect the internal network, the network-based IDS needs to see all relevant traffic traveling in and out of the network. This justifies its placement "outside" of the internal network, and closest to the ingress point as possible.

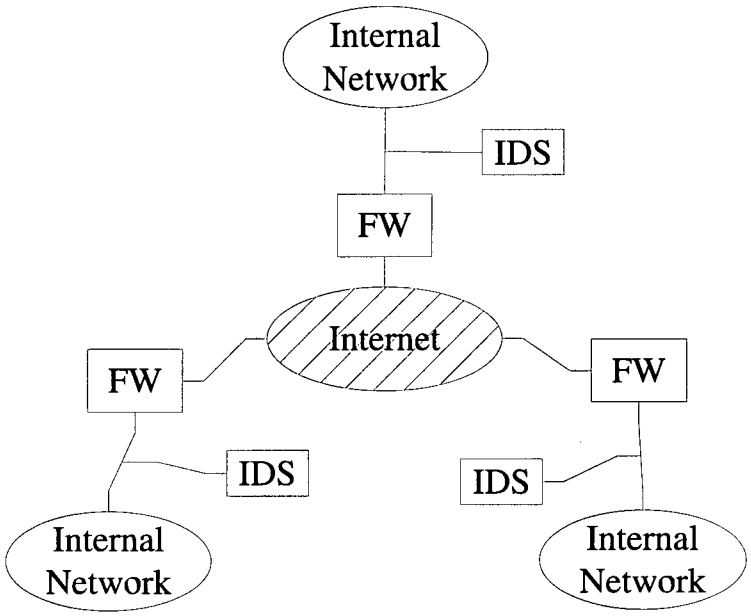

Figure 2.2: An example topology of a multi-site network-based IDS deployment.

The MS scenario (shown in Figure 2.2) represents a case in which an institution's network may be partitioned and distributed across multiple sites. Corporations with global online presence or VoIP providers are a few examples of organizations with this type of network infrastructure. The placement of the IDS behind the firewall within each partition follows the same reasoning as the SSMB case, however, each IDS is solely responsible for the network partition it protects.

2.3 Log Correlation

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 symptomatic of a single attack against multiple targets. 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.

The need for correlation is also illustrated by the deployment scenarios described in Section 2.2. Both the SSMB and MS cases have distributed IDS deployments. To find correlated events collected at each of the IDS locations, the data from each system will need to be gathered and processed.

In practice, effective correlation of results is an essential feature of modern intrusion detection systems. The conciseness and relevance of the results directly increases the effectiveness of a security expert or administrator who relies on the system, which in turn directly translates into a reduction of operating costs. A strong correlation system, therefore, does not only represent a technical advantage but can result in real financial value.

In the conventional approach to the log correlation, distributed sensor data is first gathered into a central processing location. Searching and data aggregation techniques are then applied to discover patterns within this data. The complexity and efficiency of this analysis can vary widely, and is highly dependent on each implementation. Typically, a full IDS distribution will include a detection and logging engine intended to be deployed at one or more locations, and a data aggregation and analysis tool to pool and process data from these locations and provide higher level results to a user.

The advantages of this centralized approach are its simplicity, and its relative unobtrusiveness to an existing network deployment. Network-based intrusion detection monitors can be placed in the network without affecting other hosts. Data collection and analysis can also be performed with only the additional cost of the bandwidth required to transfer the data.

This approach has significant disadvantages as well. These include:

- *Bandwidth Scalability*: The bandwidth required to collect large, distributed data sets from distributed sensors can pose a significant over-

head cost, affecting network performance.

- *Processing Scalability*: The processing capability centralized approach is limited by the computational power of the single analysis center, even though other resources may be available.
- *Analysis Delay*: In the centralized approach, logs are collected only periodically, and thus correlated results are delayed by at most one collection and analysis cycle, potentially hampering a timely and effective response.
- *Single Point of Failure*: As with any centralized system, the entire system ceases to function when the central system fails. An interruption in intrusion detection analysis tasks can result in *false negatives.*
- *Integration and Flexibility*: Existing commercial IDS's are sold and developed as standalone products, and most do not support aggregation of data between competing systems. Competing commercial IDS's have little incentive to cooperate with each other, even though the data they each collect may yield better results when combined.

2.4 Mobile Agents

The IDS framework developed for this thesis is based on a networked software technology and programming paradigm known as *mobile agents.* This section will introduce and describe this paradigm, and explain its advantages and disadvantages.

2.4.1 Definition

The mobile agent is a combination of two concepts: *mobile code* and a *software agent. Mobile code* refers to a technology that allows program code (in any form) to move from one host to another using a network link. The most common instantiation of this idea today can be seen in dynamic internet content, such as Java-based applets, Macromedia Flash based interactive

graphics, and even scripting languages embedded into web pages (JavaScript for example).

As mobile code is most often useful when it is able to move across heterogeneous environments, the two common approaches to achieving mobility are to use an intermediate object code-based system (such as Java) or use an interpreted language system (such as JavaScript, Perl, Python, or Tcl). To transport code variables transparently, mobile code systems also require infrastructure for data serialization, which is typically included in the both intermediate code-based systems and interpreted language runtimes. A platform providing both cross-platform code and data representation provides the foundation for a mobile agent system. Many mobile agent platforms have been developed, and they are too numerous to detail here (the reader is instead referred to literature available concerning each system [17, 25, 22])

The term *software agent* (or simply *agent*) has several accepted definitions, as it is used differently in the various sub-fields of computer science research. Although each community uses the word to refer to somewhat varying ideas, a common theme can be extracted: an agent is a software program that acts *autonomously* to perform actions on behalf of a user.

The combination of these two ideas forms the mobile agent. A mobile agent is a software agent built on mobile code technology, realizing a software entity that is both autonomous and mobile. A mobile agent is a software program that can move *itself* from one networked host to another, reacting to changing conditions at each host or in the network while attempting to perform some task defined by its programmer.

2.4.2 Pros and Cons of Mobile Agents

Mobile agents offer many possibilities for designing distributed systems. Past research has identified many generic situations in which mobile agents can improve performance, reduce bandwidth usage, or enable simpler system designs [15]. The major categories of these tasks are summarized below.

- Service Customization: In the mobile agent paradigm, resources may be provided by remote servers, but the service logic is often imple-

mented in the mobile agent. This implies that services can be adapted, or new services can be developed, without requiring a change to how the server provides access to its resources.

- Deployment and Maintenance: The mobile nature of the agents renders the software deployment problem trivial. Agents can deploy themselves to relevant locations, or client code can ask for logical code units on-demand.

- Autonomy and Fault Tolerance: Each agents' autonomous execution, combined with the distributed operation of multiple cooperating agents, can create what is often referred to as an "agent colony" that is highly resistant to single points of failure. A failure in part of the network only causes a subset of agents to stop execution, and communications failures (if only temporary) are handled automatically as agents continue to execute even if a communication channel becomes unavailable.

- Data Management Flexibility and Protocol Encapsulation: Because agents can bring processing logic to the location of the data, the system designer is given much greater flexibility in choosing where to store and manage data. As the movement of agents (including marshalling and de-marshalling of data) is transparent in most mobile agent systems, the system implementor is freed from having to design specialized protocols for both data transfer and processing coordination.

As with any new technology or paradigm, there are distinct disadvantages that need to be identified and considered. Listed below are the major drawbacks of the mobile agent technology:

- Requirement of an Agent Environment: Mobile agents require an operating environment similarly to how ordinary programs require an operating system. Because no operating system currently provides an agent engine as a standard feature, a system implementor must be able to access and install additional software on existing network architecture.

- Performance: As agent platforms attempt to provide a homogeneous environment over a heterogeneous network, the typical approach is to utilize virtualization technology such as the Java programming language and virtual machine. Although performance of these virtualization technologies continue to improve, they always incur a processing overhead that needs to be calculated.

- Security: Mobile agent platforms may represent a security risk when used in certain types of situations. Although several protection mechanisms exist, because mobile agent environments are intended to allow arbitrary code to arrive from a remote destination, the potential damage caused by a security breach is significant.

- Interoperability: Although standardization efforts have made some progress in defining interfaces between agent systems [33, 31], the large number of incompatible platforms remains an obstacle to wide-spread deployment.

2.4.3 Mobile Agent Security

When considering a mobile agent-based approach, the security problems of mobile agent technology must be addressed.

Research into mobile agent systems flourished in the late 1990's thanks to the explosion of the Internet and code virtualization technologies such as Java. Their widespread use, however, has been hampered by the difficulty in solving security problems related to their mobility. The most difficult of these is the *malicious host* problem, which describes the case when an agent cannot trust that host that is responsible for executing its code. Indeed, the fundamental difficulties in the problem have so far prevented any viable solutions. [1] As researchers realized the difficulty in dealing with the security issues of the mobile agent paradigm, the popularity of the paradigm declined, and systems based on it have since been few and far between. The mobile

[1] Several proposed solutions exist, such as one known as *cryptographic traces*, but their limitations make them impractical for real-world use [40, 37]

agent security problem can be solved when their application is limited to closed systems. In the case of an intrusion detection system, all the relevant network elements are owned by the same entity, and furthermore, the best practice would dictate that these elements are only accessible by their owner. In this type of context, we can guarantee the safety of mobile-agent based programs by using established encryption and digital signature techniques. Below is a brief overview of security issues related to mobile agents, and a description of how each issue is handled in the context of intrusion detection.

- *Malicious Agent*: This scenario arises when an attacker tries to send an agent that contains code not authorized by the system owner. This problem can be prevented by applying digital signature to each agent, and forcing the agent engines to verify a correct signature before execution. In an IDS scenario, all agents would be signed with a signature generated by the IDS operator, and agent hosts would be programmed only to accept agents signed by this signature.

- *Agent Tampering*: An attacker may attempt to tamper with the code of an agent as it moves through the network, causing it to execute potentially destructive code. This type of attack is similar to the *Malicious Agent* scenario, and can be dealt with by correctly applying the digital signature technique, in combination with a message digest covering the agent's code.

- *Agent Code Theft*: In certain situations it may be to an attacker's advantage if the mobile agent code can be captured and freely examined. In our context, for example, an attacker could learn about the kinds of analysis being performed by examining an analysis agent's code and use this information to perform attacks that an agent may fail to discover. To prevent this type of attack, a shared key encryption scheme, or public/private key encryption scheme can be used to protect each agent's content

- *Malicious Host*: As previously mentioned, an intrusion detection system is a closed environment. Here we make the assumption that secu-

rity mechanisms in each network element are in place, and an attacker is unable to corrupt the agent environment, in effect, preventing any of the hosts from becoming malicious.

Aside from preventative security measures, an intrusion detection for mobile agents has also been proposed [38] to perform monitoring and auditing on the Aglets mobile agent agent platform.

2.5 Mobile Agents in Intrusion Detection

Several other research projects have explore the use of stationary and mobile agents in intrusion detection systems.

The AAFID (Autonomous agents for intrusion detection) [6, 23] 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 introduced since. Extended work on this system has been reported recently as well [42]. This system realizes many of the benefits that a modular agent-based design provides, but does not explore the benefits of code mobility, there by not addressing the deployment and integration problem.

Other projects have explored the advantages of the mobility aspect of mobile agents in the context of intrusion detection. The IDA system [2] employs mobility to identify the source of attacks.

Systems which use mobile agents to model biological immune systems have been proposed [12] and elaborated [13]. In these systems, agents act in semi-random manners that mimic immune systems, in an attempt to realize fault tolerance and robustness that are found in nature. Similarly, a system modeled after the mechanism by which ants guide fellow ants to points interest provides a distributed control mechanism that does not sacrifice the fault tolerance characteristics of mobile agents.

This research, however, addresses only how the agents are coordinated and distributed, and not the actual intrusion detection tasks of the agents themselves. The Micael system [9] and the MA-IDS system [27] both use

mobile agents to aggregate distributed information related to attacks. The Sparta system [24] uses mobile agents to provide a query like functionality to reconstruct patterns of related events distributed across multiple hosts. The MAIDS project Iowa State University [18]has explored using dynamic agent composition techniques to create an array of lightweight agents to perform a full range of IDS-related tasks.

2.6 Mobile Agent Visualization

Mobile agent technology has historically proven useful for building highly distributed software, in which autonomous software agents move and perform their tasks in parallel. Mobile agents can navigate network topologies, and cooperate with other agents to carry out a collective task. However, because of the highly distributed and parallel nature of mobile agent-based systems, the task of understanding a system's behavior can be difficult. This difficulty in turn presents a challenge when the behavior of a system must be verified according to its design. For a mobile agent-based system, this involves understanding how agents are moving individually, how agents move with respect to each other, and how agents are reacting to their environment. Externally to the work presented in this thesis, a visualization tool was developed to address the difficulty of comprehending the behavior of a mobile agent system. Details of this work are presented in Appendix B.

Chapter 3

APHIDS Design and Implementation

This chapter defines an abstract framework for intrusion detection. The purpose of this framework is to allow comparisons between existing systems, as well as understand of how a APHIDS relates to these existing systems.

3.1 Abstract Architecture of an IDS

3.1.1 Finite State Machine-based Architecture

In general, an intrusion detection system can be thought of as a finite state machine, which responds to events that occur in the network and raises an alarm when a particular state in this FSM is reached.

In a system such as Snort, the finite state machine performs a function similar to a FSM-based regular expression engine. The 'events' from the network are delivered in the form of bit sequences being transferred on the network, and the FSM is organized to detect certain patterns within these bits.

A different FSM model, which has been explored by systems such as NetSTAT [21, 39], organizes the system into two components: a subsystem to monitor and generate abstract events observed on a network, and a sepa-

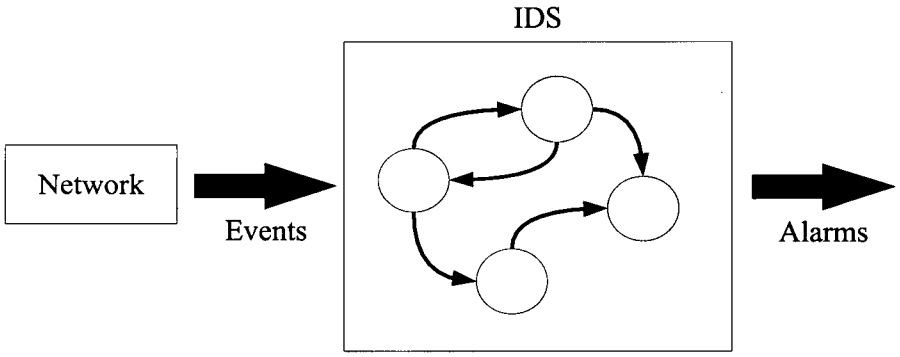

Figure 3.1: An abstract finite state machine-based model of an intrusion detection system. A network monitor component translates network activity into abstract events, which cause state transitions within the FSM. When the FSM reaches a certain state, an alarm is raised.

rate FSM-based subsystem to recognize sequences within these events. The generated events cause transitions in finite state machines built to identify specific attack patterns. This type of structure allows an IDS to identify high level event patterns rather than simple bit sequences. Figure 3.1 shows a graphical representation of this concept.

3.1.2 Mobile Agents – Distributed Dynamic Cooperating Finite State Machines

A colony of mobile agents can be considered as a distributed set of cooperating finite state machines. Each agent can send output to other agents, and can react to input from other agents.

Mobile agents have functionality that are typically not represented in FSM-based software models. Specifically, the actions a mobile agent can take during a state transition can include the creation of new FSMs (in the form of new agents) and the movement to a different host, resulting in the dynamic change of the state machine's operating environment (which potentially changes the behavior of the agent).

This dynamic nature of the mobile agent FSM model allows for a entirely

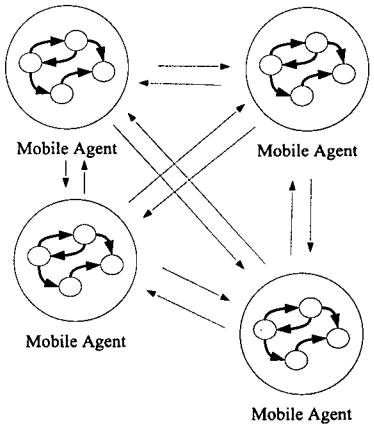

Figure 3.2: A mobile agent colony can be viewed as a distributed set of state machines that respond to input from other agents, and provide output to other agents. Agents can also create new agents and migrate from context to context.

new possibilities. The mobility and distributed coordination features should allow for novel intrusion detection system designs. However, the distributed nature of mobile agents can also lead to immense complexity requiring careful, multi-threaded, and asynchronous designs that are challenging to implement correctly and robustly. Thus, for a system to be technically successful, a practical balance between flexibility, dynamicness, usability, and manageability must be achieved.

3.1.3 Intrusion Detection Process Model

Most existing intrusion detections perform the same general process but employ different techniques at different phases within this process. As APHIDS aims to be a generic framework on which task-specific intrusion detection system can be built, it is useful to define a conceptual process model for intrusion detection.

The process model used for APHIDS defines three distinct phases, with a one-way progression through each phase. The three phases are described

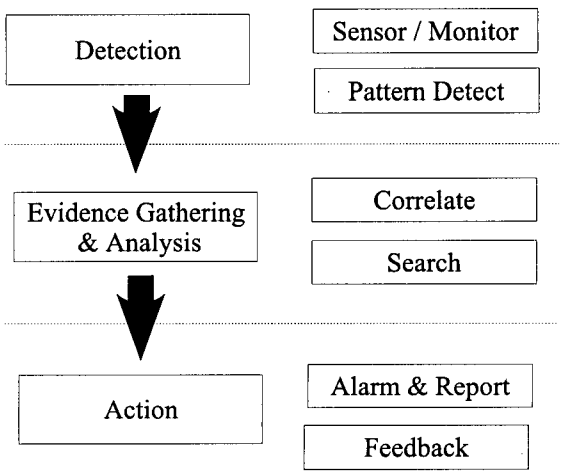

Figure 3.3: APHIDS defines a model for the intrusion detection process containing three phases. The process starts with the detection of an event, which is followed by an analysis and search phase, and is concluded with an action phase in which any appropriate action can be taken.

below, and are also show in in Figure 3.3

1. Detection: This phase is responsible for the monitoring and detection of attacks. This can involve monitoring for simple single events, or employing mechanisms to detect patterns of events.

2. Evidence Gathering and Analysis: This step attempts to collect information regarding an attack detected by the Detection phase. This includes searching for and correlating distributed data captured by different monitoring systems.

3. Action: Once analysis is complete, an action needs to be taken. This would typically communicate correlation and analysis results to the user, but the a system could also use the results, for example, to intelligently re-configure itself.

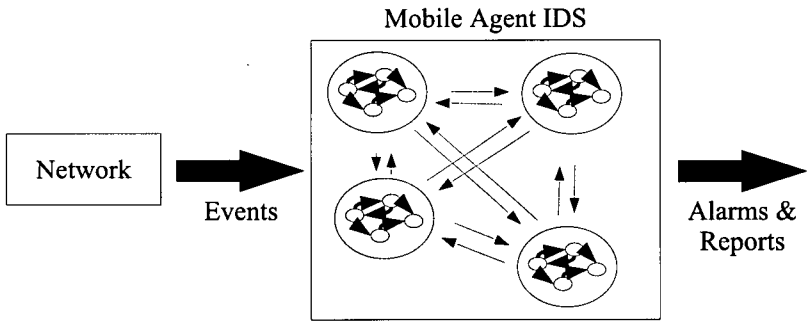

Figure 3.4: A mobile agent-based IDS can be seen as a state-transition based system, with a set of distributed, interacting extended state machines.

3.2 APHIDS System Design

The APHIDS architecture can be seen as a combination of the view of an IDS as a finite state machine, and the mobile agent distributed FSM model. Co-operating agents respond to network events and generate alarms and reports based on these events. The APHIDS model, however, represents only one of the many possible organizations of distributed mobile agents to achieve the overall goal of intrusion detection.

The organization of agents in the APHIDS system is guided by the goal to implement the intrusion detection process model described in Section 3.1.3. Each phase in the process model is mapped to distinct subsystems, each of which are implemented by specialized mobile agents. While the application process model may restrict the domain of possibilities offered by the mobile agent paradigm, the process' clear structure aids in the definition and management of relationships between the various cooperating agents.

The detection phase is implemented by the *trigger agent*. The role of the trigger agent is to monitor an external system for suspicious behavior and report any observed events to the *correlation agent*, which is responsible for the next step of the process. The event that is observed and reported by a trigger agent is named a *trigger event*. As trigger agents can be created to detect any type of event, a trigger event only refers abstractly to the event

reported by the trigger agent.

The correlation agent, once it has received information regarding a trigger event, will perform any analysis tasks that are required to obtain more information regarding this event. Each correlation agent is programmed with a *correlation routine* which defines its behavior when responding to a trigger event. This mechanism is described in detail in Section 3.4.1. Finally the correlation routine may require an *action agent* be launched based on the results of the analysis.

This action agent implements the final step of the process. However, to allow for flexibility, the design does not specify the exact behavior of the action agent. An action agent can be defined to perform any task, allowing for the framework to adapt to a greater variety of situations.

3.3 Implementation Objectives

Several goals were defined for the prototype implementation of the APHIDS system. This project aimed to create a novel IDS architecture with novel functionality but at the same time address some of the limitations of current IDS systems. Specifically, the APHIDS system was designed to address the limitations of the centralized analysis approach (detailed in Section 2.3). Being a mobile agent-based system, the implementation also aimed to realize all of the potential performance benefits of the mobile agent paradigm. Specifically this required that:

- Analysis is distributed, allowing for efficient usage of distributed computing resources. Bandwidth overhead is kept to a minimum.
- Analysis tasks do not only combine similar types of information from distributed homogeneous sources, but also different types of information of distributed heterogeneous sources. The system is able to integrate and process data generated by existing intrusion detection and monitoring technologies.
- Alarms are raised as soon as possible after the detection of suspicious behavior.

As APHIDS aims to be a generic intrusion detection platform, it must provide meaningful abstractions and useful modular design. Modularity in the design was focused on achieving the following features:

- Integration: the ability to integrate existing intrusion detection systems and techniques
- Extensibility: the ability to improve and extend the system feature set without requiring fundamental changes to its architecture. This includes the ability to adapt to new attack patterns and implement new analysis algorithms.
- Programmability and Automation: the system should provide a mechanism for a user to re-use existing components or customize existing analysis procedures. The system should provide a mechanism to allow security experts to automate analysis procedures that would otherwise be performed manually.

3.4 Implementation Details

3.4.1 Programmable Correlation: Correlation Routines

Abstractly, a *correlation routine* refers to the list of analysis and search procedures that the IDS system needs to perform in the correlation phase when responding to a particular detected event. Different correlation routines describe different analysis procedures for different trigger events. For the system to be able to respond to a large variety of events, it must be able to allow a diverse set of routines to be "enabled" simultaneously.

APHIDS accomplishes this requirement by delegating one correlation agent to handle each correlation routine. A correlation routine is described using an active object, a CorrelationObject, which provides functions that implement the analysis logic. When a correlation agent is initialized, a name of a CorrelationObject is passed as a parameter. The correlation agent then instantiates this object and uses it to perform correlation procedures.

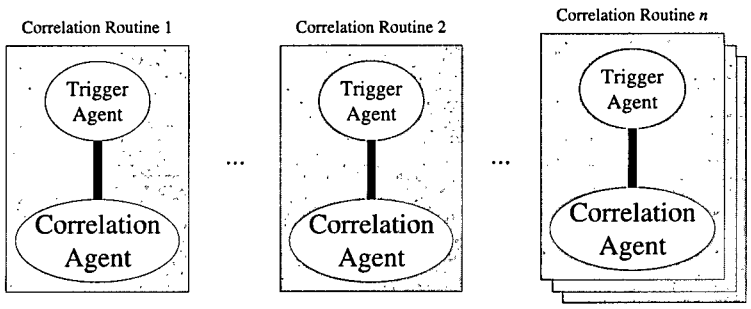

Figure 3.5: Each correlation routine is realized by a separate set of CorrelationAgent and Task Agents.

To perform the distributed search often required by correlation step, a separate type of agent is provided, known as a *task agent.* These specialized agents implement specific distributed search routines. For example, a task agent may be programmed to migrate to a remote host and search through its webserver's log file and return entries matching certain criteria. These task agents are loosely coupled to allow different CorrelationObject's to re-use their functionality, simplifying the development of a complex CorrelationObject.

A single correlation routine cannot possibly handle all types of trigger events. To reduce the complexity of the correlation routine, each CorrelationObject is responsible for the deployment of its own trigger agent. This mechanism allows the correlation routine to control exactly which types of events it will receive, and implement the appropriate analysis logic for those limited sets of events.

In practice, this results in one correlation and trigger agent pair (as well as short-lived task agents and action agents) to be instantiated separately for each enabled correlation routine. This effect is illustrated in Figure 3.5.

It is often the case that several trigger events are delivered to a correlation agent in rapid succession. Requiring each CorrelationObject to implement its own multi-threaded analysis procedure to handle this situation could make CorrelationObject development error-prone and cumbersome. There-

fore, the correlation agent is designed to create multiple threads of analysis, each thread with its own copy of the CorrelationObject, its own variable name space, and its own execution state, so that several trigger events can be handled simultaneously. This functionality is hidden behind the CorrelationObject API, to allow CorrelationObjects to be developed with a simple single-threaded design.

3.4.2 Customizing Trigger, Task, and Action Agents

Trigger, task, and action agents have simpler designs, and do not allow for customization by parameterized active objects. APHIDS simply provides abstract superclasses with a small set of methods that need to implemented by subclasses. Allowing the direct extension of these classes provides for greater flexibility in their implementation, as the subclasses will be able to access all the features provided by the mobile agent platform (as a result of the new implementation deriving from a mobile agent base class).

Detailed descriptions of the methods that need to be implemented by subclasses of these agent types, as well a description of the CorrelationObject API, can be found in Appendix A.

3.4.3 System and User Interface Agents

To manage the potentially large number of correlation agents and trigger agents, a separate management agent (known as the system agent) was implemented. This agent provides a high level interface to query available correlation routines and to enable and disable specific routines. The system agent also maintains a list of references to all operating correlation agents[1] in the system, allowing for centralized system management functions such as system shutdown, in which all active agents must suspend their processing and free their resources.

An UI agent is also provided to present a simple user interface on the

[1]Correlation agents are responsible for deploying their own trigger agents, and also maintain a reference to each trigger agent. Therefore, the correlation agent is considered the single point of contact for the set of agents implementing one correlation routine.

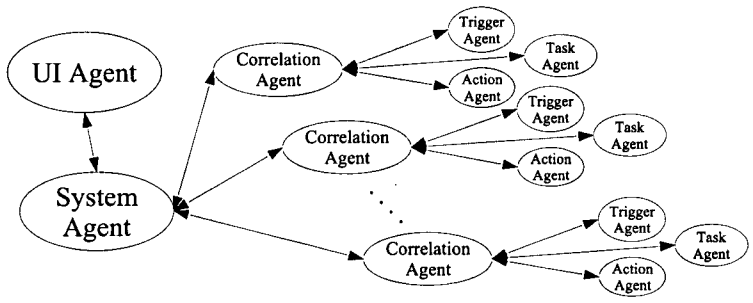

Figure 3.6: Depicted are the relationships between the various agents found in the APHIDS system. The System Agent is responsible for tracking and managing Correlation Agents, each of which are responsible for their own Trigger, Task, and Action Agents. User commands are performed through requests made to the System Agent.

"console" host[2]. This agent translates user commands into procedure calls for the system agent, allowing the user to interact with the system.

A high level view of the relationships between the APHIDS agents is provided in Figure 3.6.

3.4.4 Deployment

From a deployment perspective, the structure of our system is trivial. It requires only the placement of an agent engine at every relevant location, including network monitoring devices, web servers and other service-providing hosts, and more conventional IDS implementations as shown in Figure 3.7. APHIDS is realized as a distributed layer which operates on top of a set of distributed agent engines.

Each agent engine location is considered a potential data source. In the case of engines being placed alongside an existing IDS, the engine should be installed in a manner that allows for efficient access to the data generated by that IDS deployment. The architecture also requires the existence of one or

[2]The UI agent is a mobile agent itself, and uses only standard inter-agent communication mechanisms to communicate with the system agent, potentially allowing for a mobile GUI in the future.

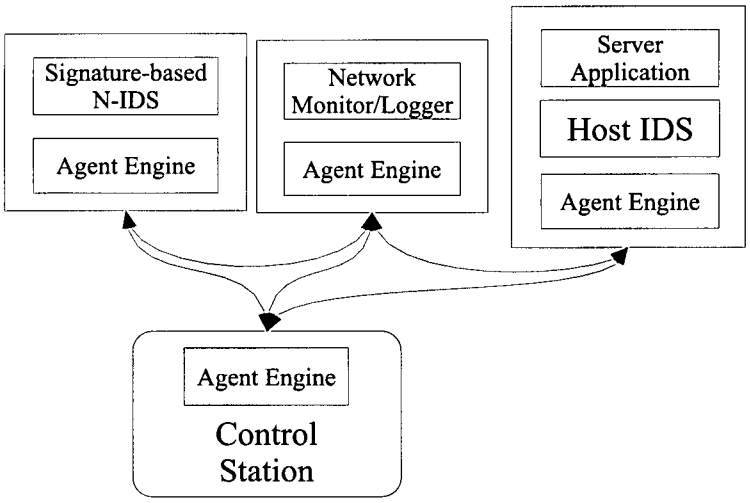

Figure 3.7: Depicted is a generalized view of our system architecture. Each data source (network sensor, network-IDS, host-IDS, etc.) runs an agent engine. Agents can move to and from each engine (as depicted by the arrows) and perform analysis tasks

more system consoles, used for system configuration and display of analysis results.

All of the system functionality is implemented above the mobile agent platform layer. This design allows the access of network components in a uniform and efficient manner, and allows all of the system components to take advantage of the benefits provided by the agent platform.

3.4.5 System Processing Flows

This section describes several common execution flows through the system. These descriptions aim to illustrate the functional relationship between the various agents in the APHIDS system.

- System Initialization: The system is initialized by the user by loading the System Agent using the Grasshopper user interface. Upon startup, the system agent initializes its data structures, deploys an UI agent, and establishes communication with the new agent.

- UI Commands: The command-line based operations read in by the UI agent are parsed and mapped to corresponding RMI calls to the System Agent. Any return values from these calls are converted by the UI agent into text form and displayed in the UI's text output area.

- Routine Activation: When an "enable" command is entered into the UI, the UI agent will use the first argument to this command to cause a correlation routine activation. This argument (the class name of the Correlation Object implementing the desired routine), is passed to the System Agent using an RMI call. The System Agent then verifies the validity of the CorrelationObject class name and requests the underlying Grasshopper system to create a Correlation Agent instance. The Correlation Agent is passed the Correlation Object class name as a parameter, which it uses to initialize the CorrelationObject using Grasshopper's class loader implementation. The Correlation Agent then calls a method provided by the CorrelationObject to deploy the trigger agent needed by this routine. Using the information returned by

this method, the Correlation Agent configures the plumbing required to pass incoming trigger event information to the CorrelationObject. They System Agent, upon returning from the request for the Correlation Agent's creation, creates an RMI proxy object for the new Correlation Agent, and stores this object for later use.

- Routine Shutdown: A routine (and corresponding Correlation Agent) is disabled by the "disable" command through the user interface. When the System Agent receives the corresponding RMI call, it will locate the RMI proxy object for the correlation agent handling the specified correlation routine. Using the proxy, the System Agent will ask the Correlation Agent to cease its operation. This call returns immediately, as the Correlation Agent performs its shutdown operation in a separate, newly spawned thread. This thread attempts to contact the trigger agent to request its shutdown, then performs any necessary cleanup, and finally notifies the System Agent of its own shutdown just prior to removing itself. The System Agent, upon receiving this shutdown notification, discards its stored RMI proxy for the correlation agent, and records the correlation routine as disabled.

- System Shutdown: A system shutdown is initiated by the user by clicking on the shutdown button provided by the GUI. The system agent performs the same "Routine Shutdown" procedure described above, for each enabled routine, then performs its own shutdown routine.

- Analysis Cycle: An analysis cycle is started by the trigger agent. The trigger agent performs an RMI call on a correlation agent, communicating the details of the detected trigger event. The correlation agent places this information in a queue for incoming trigger events. An free analysis thread (started by the correlation agent during initialization) obtains the new trigger event off the queue, then creates its own copy of the CorrelationObject, initializes it with the trigger event information, and calls a pre-defined method to perform the programmed correlation routine. During this routine's execution, the Correlation Object can

request the execution of task agents and action agents through methods in its API. Upon such a request, control is transferred temporarily to the correlation agent, which requests the underlying agent system to create a new agent instance. This new agent is provided with information regarding its creator, as well as information to allow the correlation to associate incoming data with a particular executing routine. The new agent is then launched (which occurs asynchronously) and control is returned to the Correlation Object. The analysis cycle completes whenever the correlation routine finishes execution. The launching of an action agent is not mandatory, allowing the correlation routine to "ignore" certain events if appropriate.

3.4.6 Implementation Environment

The prototype system was developed over the Grasshopper mobile agent platform [3]. The Grasshopper system is a Java-based, freely available, mature mobile agent platform. It requires only a standard Java (Java 2 Standard Edition 1.4 was used for development) virtual machine platform.

This platform was chosen due to its relative completeness (both in terms of feature set as well as documentation), its emphasis on implementing the mobile agent-related standards, and its simple yet powerful structure. Grasshopper provides a built in RMI mechanism that is capable of both synchronous and asynchronous calls between agents, as well as a directory service and other management features. Grasshopper serves as a simple platform for developers familiar with Java to create complex agent systems.

Development and testing was performed on a locally available cluster (details available in Table 4.1). Although the design theoretically scales to a large number of hosts, its operation has been verified only on this group of 6 hosts.

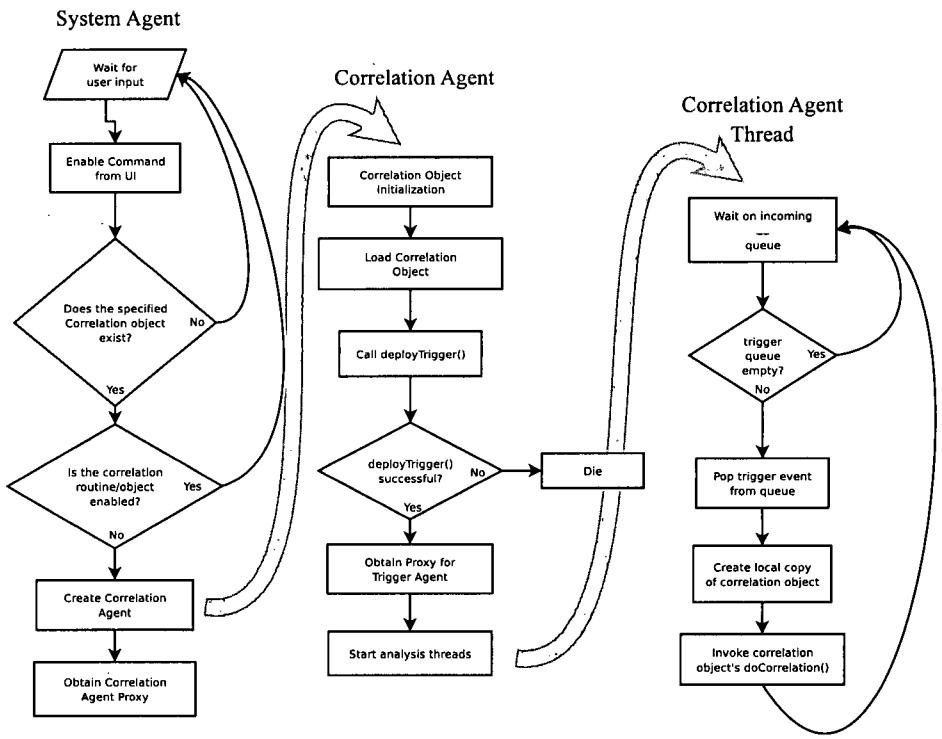

Figure 3.8: This flowchart describes the processing flow between the system agent and correlation agent upon a request that a correlation routine be enabled.

Chapter 4

Evaluation

4.1 Theoretical Analysis

This section presents mathematical analysis intended to identify and characterize situations in which the mobile agent approach is advantageous. The system is compared to the centralized correlation approach, as this is the common approach used by most existing systems, and many implementation details of other mobile agent based systems are unavailable. As the behavior of an IDS is highly dependent on its environment as well as in its techniques, several simplifying assumptions are made in an attempt to analyze the average case.

4.1.1 Bandwidth Usage

To analyze the bandwidth costs incurred by the centralized approach and the APHIDS approach, some common variables first need to be defined.

- N_i: the total number of intrusions in a given time period
- S_{agent}: the size of an average agent in bytes
- S_i: the total size of all the data (distributed among one or more locations) related to one intrusion
- N_{hosts}: the number of hosts in the network

- S_{data}: the total size of data collected at one host by all monitoring systems

The bandwidth consumed by one incident, with analysis by mobile agents can then be defined as:

$$B_{single} = N_{hosts} * S_{agent} + S_i \tag{4.1}$$

Here, it is assumed that for each incident, one task agent needs to be sent to each host. Also, each task agent will need to send back the relevant data regarding that incident, which is captured by S_i.

The bandwidth consumed by N_i incidents is then:

$$B_{total} = B_{single} * N_i \tag{4.2}$$

In the central analysis scheme, the total amount of bandwidth used is simply the size of all the data (relevant or not) at each host times the number of hosts.

$$B_{central} = S_{data} * N_{hosts} \tag{4.3}$$

To justify the case that the mobile agent approach uses less bandwidth, the system must satisfy the following requirement:

$$B_{total} \leq B_{central}$$

Substituting in equations 4.1, 4.2, and 4.3, results in:

$$\begin{aligned} N_i * N_{hosts} * S_{agent} + N_i * S_i &\leq N_{hosts} * S_{data} \\ \frac{N_i * N_{hosts} * S_{agent}}{N_{hosts} * S_{data}} + \frac{N_i * S_i}{N_{hosts} * S_{data}} &\leq 1 \\ \frac{N_i * S_{agent}}{S_{data}} + \frac{N_i * S_i}{N_{hosts} * S_{data}} &\leq 1 \end{aligned}$$

This equation approximately defines the fundamental requirement for the mobile agent approach to save bandwidth costs over the centralized one. Although further algebra will not be able to combine the two terms on the left of the inequality, each term can be analyzed individually. As the sum of the two terms must be less than one, this requires that at minimum, each term must be individually less than one. These simplified conditions have several implications. First:

$$\frac{N_i * S_{agent}}{S_{data}} \leq 1$$

This result indicates that the number of incidents times the average size of an agent, must be smaller than the size of the data stored at each host. In other words, if there are a very large number of incidents, the overhead of transferring each analysis agent during each incident cannot become larger than the cost of transferring the entirety of the data the agent wishes to analyze. This component expresses the fundamental requirement of the *remote evaluation* technique.

$$\frac{N_i * S_i}{N_{hosts} * S_{data}} \leq 1$$

The second term shows another requirement. For a group of agents to perform collective analysis, they are going to need to transfer the data relevant to each incident across the network at some point in the analysis process. The inequality implies that as the number of incidents increase, the size of the "relevant" data pertaining to an incident must remain small when compared to the total of all the data on every host. This term without the N_i factor can be thought of as a measure of *filtering effectiveness*:

$$\frac{S_i}{N_{hosts} * S_{data}}$$

This ratio characterizes the degree to which the size of data that is "relevant" to a single incident is smaller than the total data collected by every system. Our assumption is that a single incident relates only to a extremely

small subset of the entire data set, as numerous unrelated incidents can be recorded over a period of time. Surely, if most of the captured data is relevant to all events, then repeatedly transferring it across a network will be relatively inefficient.

4.1.2 Computation

The argument for processing scalability can be made by examining the computational work done per host involved in a network. A situation in which, on average, a single host performs less processing work can be interpreted as achieving better distribution and being more scalable.

Below are several variables defined for the starting point of the analysis:

- $N_{h,i}$: Number of hosts containing data related to incident i
- $N_{incidents}$: Number of incidents
- W_i: Work required for incident i

The total processing load $W_{total}$to process all $N_{incidents}$incidents is:

$$W_{total} = \sum_{i=1}^{N_{incidents}} W_i$$

In the central analysis case, all work is done at one host (or a dedicated cluster). Let N_{cpu} be the number CPUs available to the analysis center, then the average work per CPU for the central case would be

$$\frac{W_{total}}{N_{cpu}}$$

In the distributed mobile agent case, the number of hosts involved in each incident differs according to the incident, and thus the workload per host can vary greatly. The average case can be analyzed, however.

Let $W_{i,d}$be the amount of total work done in the distributed case for one incident i. $W_{i,d}$can be approximated by:

$$W_{i,d} = W_i + V(N_{h,i})$$

Where W_i is the work done per incident, and $V(N_{h,i})$ is the overhead work (a function of the number hosts involved) to perform the work in a distributed manner.

Assuming the work for one incident is distributed evenly between the hosts involved with it, we can define $W_{i,h}$, the amount of work done for one host for incident i:

$$W_{i,h} = \frac{W_{i,d}}{N_{h,i}} = \frac{W_i + V(N_{h,i})}{N_{h,i}}$$

Let $P(i, H)$ be the probability that a host H is involved in the analysis work for incident i.

The total work done by a host H (denoted by W_H) then is:

$$\begin{aligned} \sum_{i=1}^{N_i} P(i,H) * W_{i,h} &= W_H \\ \sum_{i=1}^{N_i} P(i,H) * \frac{W_i + V(N_{h,i})}{N_{h,i}} &= \end{aligned}$$

For the sake of aggregate analysis, this analysis considers the average values P' and N' for $P(i, H)$ and $N_{h,i}$. Specifically, we define the following:

$$\begin{aligned} P' &= \sum_{i=1}^{N_i} P(i) \\ N' &= \sum_{i=1}^{N_i} N_{h,i} \end{aligned}$$

Treating $P(i, H)$ and $N_{h,i}$ as constants and replacing them with P' and N' allows their removal from the summation and the following simplification:

$$\frac{P'}{N'}(\sum_{i=1}^{N_i} W_i + V(N_{h,i}))$$

$$\frac{P'}{N'}(W_{total} + \sum_{i=1}^{N_i} V(N_{h,i}))$$

The term $\sum_{i=1}^{N_i} V(N_{h,i})$ is simply the total overhead work required for all the incidents, and will be denoted as V_{total}. This approximate value of $W_H$must be less then the value for the centralized case: $\frac{W_{total}}{N_{cpu}}$.The final inequality then becomes:

$$\frac{P'}{N'}(W_{total} + V_{total}) < \frac{W_{total}}{N_{cpu}}$$

The term P' can be thought of as the "involvement factor", or the likeliness that a host will be involved in a particular incident. A value of one indicates that for every incident, a host is involved in part of the processing. As each incident typically involves some subset of all the available machines (and rarely every single machine), this value will in practice be less than one. Hosts that are more "central" to the analysis procedures will have higher P' values, and those less involved will have lower values.

The value of N' indicates the average number of hosts involved with an incident. If more hosts are involved with an incident, then from the point of view of a single host, the probability of being involved in an incident increases. In this sense, P' and N' are related: as N' increases, P' is likely to increase as well.

4.1.3 Latency

In the centralized approach, the maximum time between an incident occurrence and a report is one collection cycle period T_c. In the APHIDS design, the maximum time needed to generate a report is related only to the processing time required for the analysis of a single incident T_{single}.

In order to improve upon the centralized approach, the following condi-

tion needs to be satisfied

$$T_{single} < T_c$$

If $T_c$is set to a very small value, the latency benefits of the mobile agent paradigm can be negated. However, based on a reasonable estimate of T_{single} being on the order of a few seconds, such a low T_c may cause unwanted side effects in most centralized designs. Such frequent polling of many data sources will likely cause significant processing and bandwidth overhead. In general, it is likely safe to assume that T_{single}will be significantly smaller than $T_c$in the vast majority of deployments.

4.2 Empirical Analysis

In an effort to quantify some aspects of the APHIDS prototype implementation, some measurements were performed. However, quantitative evaluation of intrusion detection systems remains a difficult problem as confirmed by the numerous approaches and critiques described in the literature [1, 10, 28, 30, 29]. The fundamental difficultly arises from the impossibility of defining a standard test scenario, as network environments can vary in many dimensions. The measurements performed for this research were limited to those that could be easily reproduced, and those that would serve as a base for comparison when future improvements are incorporated.

4.2.1 Test Environment

For development and experimentation purposes, a small cluster was created to simulate a plausible deployment scenario. A graphical description of this cluster is provided in Figure 4.1.

The cluster consists of five hosts: A system console for the APHIDS system, a host running the Snort IDS, a host providing an SSH service, a host providing an HTTP service, and finally an attacker host. Although in a real-world deployment, a firewall would be installed between the attacker

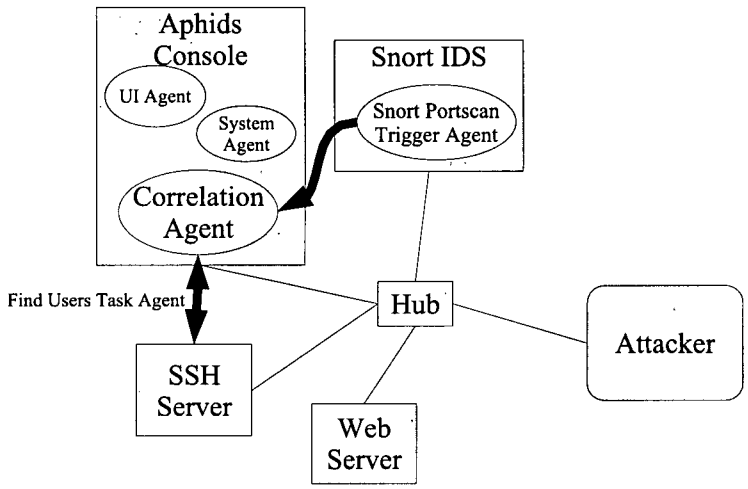

Figure 4.1: The test environment and deployment of APHIDS

Network Element	Description
Aphids Console	1.0 GHz AMD Athlon. 256MB RAM
Snort IDS	Dual 200MHz Pentium-Pro. 160MB RAM
SSH Server	200MHz Pentium-Pro. 160MB RAM
Web Server	Dual 200MHz Pentium-Pro. 160MB RAM
Attacker	200MHz Pentium. 96MB RAM
Hub	3com LinkBuilder 10BTi 10Mbps ethernet hub

Table 4.1: A summary of the hardware features of the test environment

host and the rest of the hosts, it was omitted as it would not have any effect on the measurements taken for this study.

The hardware specifications for these hosts are summarized in Table 4.1. All hosts except the console system run the Debian Linux 3.0 (Woody) distribution. The console hosts run the Gentoo Linux distribution. The Sun Java 2 Standard Edition SDK version 1.4.2., and the Grasshopper Mobile Agent Environment version 2.2.4 were installed on all the hosts.

4.2.2 Test Correlation Routine

Several software components were developed to simulate an analysis routine that correlates port scan attacks to user logins. A Trigger Agent was developed to monitor the output of the Snort IDS system (SnortPortscanTrigger), generating a notification whenever the IDS recorded a SCAN type event. In response the CorrelationObject was programmed to launch a task agent (FindUsersFromHost) to travel to the SSH host to search for login attempts from the origin of the port scan by reading through the host's authentication log file (/var/log/auth.log). With the results of a search, an Action Agent (ConsoleMessageAction) was written to contact the system agent and report the results of the analysis to be displayed by the UI Agent.

The detailed steps of this analysis procedure are described below:

1. SnortPortscanTrigger agent detects SCAN type event in Snort IDS log.
2. Trigger Agent reports information about event to Correlation Agent
3. CorrelationObject in response launches FindUsersFromHost to find login attempts on SSH server
4. FindUsersFromHost agent initializes, then migrates to SSH Server
5. FindUsersFromHost agent arrives at SSH server. Parses authentication log file, and collects entries related to SSH login attempts from attacker host.
6. FindUsersFromHost agent sends relevant log entries back to Correlation Agent
7. Correlation Object launches ConsoleMessageAction to display details of port scan event along with discovered related SSH login attempts.
8. ConsoleMessageAction agent initializes, contacts SystemAgent, and requests display of alert message.

4.2.3 Bandwidth Usage Analysis

This analysis attempts to establish the total bandwidth required for processing of one incident using the test correlation routine described in Section 4.2.2.

4.2.3.1 Experimental Procedure

The Grasshopper Mobile Agent system sends all agent related traffic through the TCP port 7000. To measure the total bandwidth usage caused by a port scan incident, the system was first initialized to monitor for the port scan, and simultaneously a packet capture program (Ethereal, which uses the libpcap library) was started to capture all packets traveling between involved hosts. The attacker host was then used to initiate a known number of port scans using the *nmap* tool. The packet capture was continued until all the analysis related to the launched port scans were completed. The captured packets were then filtered by destination port (in this case, port 7000, the port used by Grasshopper for all inter-host transfers), and the total size of this subset of the packet data was recorded.

In the detailed description of the test routine shown in Section 4.2.2, the data captured by this experiment corresponds to network transfers during steps 2, 4, and 6. Steps 3 and 7 do not cause network transfers as the newly created agents initialize on the same host as the agent requesting the creation. Likewise step 8 does not result in network transfers as the System Agent and the ConsoleMessageAction agent reside on the same physical host. Steps 1 and 5 involve only parsing of local files, which requires no interaction with the network.

4.2.3.2 Results

The procedure was repeated, each time with a different number of attack incidents. The results of this experiment are shown in Figure 4.2. The linear relationship between the bandwidth consumed and the number of incidents is consistent with the lack of caching of any sort within the system. This

measurement can be used as a baseline for all future bandwidth-saving improvements to the system (see Section 5 for ideas).

The actual amount of bandwidth used per port scan amounted to roughly 36 kilobytes. This number can be attributed to several factors:

- Each invocation of *nmap* was recorded by the Snort system as three different incidents (scans on three different ports). Thus three trigger agents were generated, and accordingly three task agents were launched. A per-Snort-event analysis would yield roughly 12 kilobytes per event.
- The size of the migrating task agent was 3509 bytes. The task agent migrates once for each analysis cycle, resulting in roughly 10 kilobytes of transfers for one port scan.
- The environment was created so that the task agent would find four entries in the SSH Server's authentication log. Each task agent invocation requires the content of these four lines to be transferred back to the correlation agent.

4.2.4 Latency Analysis

This experiment aims to measure the variation in reporting latency (the time between when an event is first detected and when it is reported). While the reporting latency of the APHIDS system should be far below any typically configured centralized analysis system, a measure of latency also gives insight into the performance characteristics of the system.

4.2.4.1 Experimental Procedure

To enable precise latency measurements, the trigger and action agents of our test routine were instrumented to record time stamps using the Java system libraries. The trigger agent was modified to record a time stamp as soon as it had detected a new matching entry in the Snort output log file. The action agent was modified to record the time just before it requested an alert

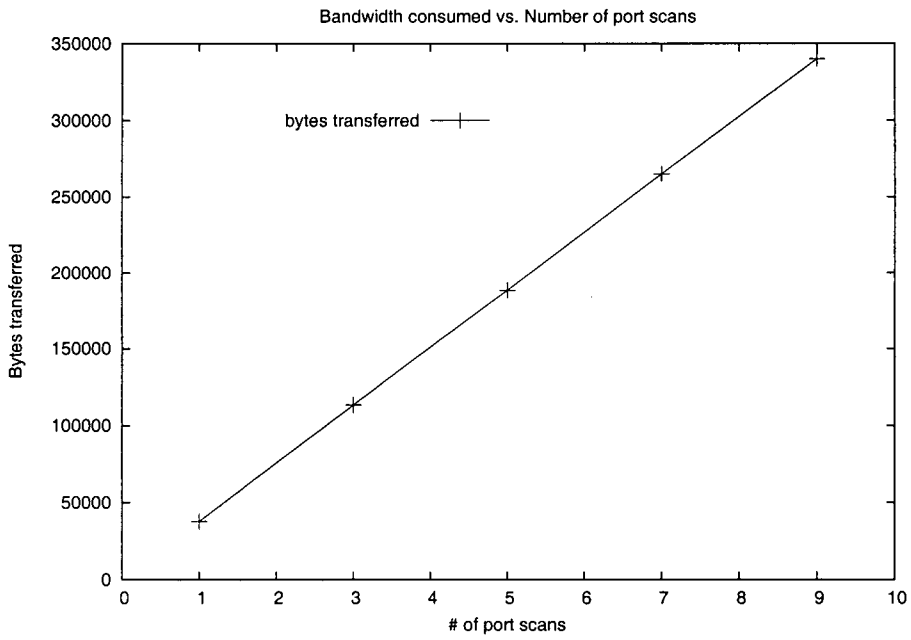

Figure 4.2: Results of the bandwidth usage measurement. The bandwidth used by the APHIDS system grows linearly with the number of simulated attack attempts

message be displayed on the system console. The difference between these two timestamps was calculated and also printed out to the system console.

As the trigger agent and action agent reside on different hosts when recording the times, an NTP (network time protocol) server and client were used to synchronize the system clocks of the two hosts. Although the NTP software does not guarantee perfect synchronization between the two clocks, any remaining clock drift would be of relatively small magnitude, and would affect all samples in the same way.

To simulate various loads, a script was created on the attacking host to invoke the *nmap* command 20 times at a different intervals. Seven different intervals were attempted (10, 7, 5, 4, 3, 2, and 1 second(s)).

The correlation agent was configured to allow a maximum of five correlation instances to occur simultaneously. This limit was chosen partially due to the relatively low capability of the test environment hardware [1], as well as instances of instability experienced within the mobile agent platform when a large number of agents were enabled simultaneously.

4.2.4.2 Results

The results from this experiment are plotted in Figure 4.3. One sample represents one reported trigger event (which equates to one complete execution of the test routine). As one *nmap* resulted in three detected events, there are exactly 60 samples per vertical group.

As the results show, as the interval between port scans decreases (causing a greater system load) the variation in analysis latency increases greatly. This data indicates an approximation of the maximum constant throughput of the prototype system. In this case, when the interval falls to 3 seconds or less, the latency increases due to the inability of the processing threads to 'keep up' with the incoming trigger events. As trigger events are queued when no free processing threads are available, the results for intervals of three seconds or less clearly indicate a lengthening of this queue. Likewise, the vast majority of samples for intervals greater than four seconds have very small

[1] The situation is exacerbated by the relatively resource-demanding Java virtual machine.

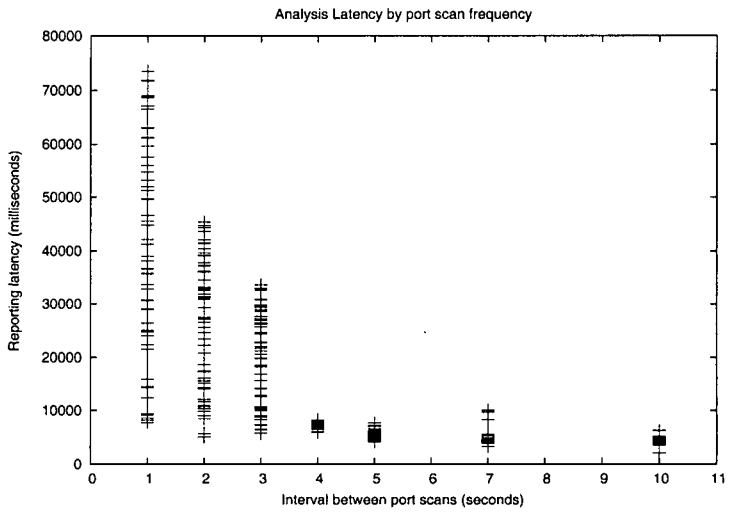

Figure 4.3: Results of the measurements of reporting latency. Each mark represents one reported event. As the delay between port scans is reduced, the variation and upper bound of the reporting latency increases

variation, indicating that at these loads, the system is able to always react immediately and consistently to new incoming trigger event notifications.

An important concern for all intrusion detection systems is that the IDS itself may become a target of an attack. Although the results of this experiment are highly dependent on the correlation routine and the operating environment, the existence of a processing capacity indicates that if attackers were to generate repeated attacks at a frequency greater than this threshold, the IDS could become overloaded, and in effect, disabled.

4.3 Qualitative Analysis

4.3.1 Processing Overhead

The decision to create a correlation agent for every enabled correlation routine has several ramifications:

- When a large number of routines are enabled, an equally large number of distinct agents will operate in the system. In the Grasshopper platform, each agent's execution is performed in a separate thread. With a large number of agents, the thread switching overhead in the Java virtual machine may become a significant component of the total overhead.

- As a correlation agent also requires memory, the enabling of a significant number of correlation agents may result in a large memory overhead.

Several alternative approaches were also considered. One involved creating only one correlation agent for the entire system, and having this agent manage all correlation routines. This approach has the potential advantage of reducing memory footprint (as only one correlation agent is required) and also has the flexibility of globally limiting the number of processing threads required by the system.[2] However, such a system would require a scheduler-like mechanism to decide in what order to execute routines, adding to the complexity of the correlation agent.

Another involves creating several correlation agents, each capable of handling multiple routines, and balance the routine distribution between these agents. By tuning the ratio of routines to agents, a trade off can be achieved between memory usage and thread explosion. This approach, however, also requires a scheduler mechanism to be designed, implemented, and studied.

In designing the APHIDS, the one-to-one routine to agent mapping was chosen for several reasons:

- Simplicity of Design: A one to one mapping requires that a correlation agent only be concerned with the state of one routine, resulting in a simpler design and easier verification.

- Future Flexibility: The design allows for a future extension in which

[2]This can be implemented by using a thread pool with a maximum number of threads to limit the number of concurrently processed correlation routines.

a colony of correlation agents could independently load balance themselves across several processing resources.

- Fault Tolerance, Separation: Having an entire correlation agent available for each routine ensures that if a particular routine fails (and in the process disables its correlation agent), no other routines are affected.

4.3.2 Extensibility

The modular APHIDS design allows the system to be extended in several important ways. Each form of modularity and its benefits are summarized below:

- Decoupled Agent Subsystems: The separation of the three stages of the intrusion detection process, and the definition of generic interfaces between them allows for entirely new subsystems to be developed and implemented, rendering APHIDS a useful platform for experimentation.

- Correlation Routine: The correlation routine abstraction, and the CorrelationObject mechanism allows developers of correlation routines to take advantage of the APHIDS infrastructure without the need to directly manipulate underlying agent structures and systems. A simple API makes development of new routines quick and less error-prone. Likewise, the abstraction also allows for arbitrarily complex, multithreaded correlation routines to be implemented.

- Agent Platform: The reliance on a mobile agent platform greatly simplifies deployment problems. Newly developed agents and correlation routines can be deployed immediately, requiring no restarting or reconfiguration of existing enabled infrastructure.

- Data Sources: By delegating interaction with external systems to Trigger, Task, and Action agents, correlation routines can be developed to integrate data from any source, potentially combining the data of all available monitoring systems.

4.3.3 Advantages of Mobile Agents

By utilizing mobile agent technology, APHIDS is able to provide several features that are unavailable, or difficult to implement using the standard client-server approach.

- Rapid Deployment: Although the current work assumes that all agent engine hosts are considered trusted (implying that software deployment itself is not a problem), when the number of such hosts grows large, manual deployment of new analysis software can become costly. Mobile agents remove the deployment problem, allowing for new analysis or monitoring code to be integrated on-the-fly.

- Programmable Remote Evaluation: Mobile agents do not need to rely only on services provided by the remote host (whether it be a monitoring host or a host containing relevant data). Rather, they can bring service logic to the data, allowing a programmer to utilize data in new ways without requiring a system configuration change. The remote evaluation model also allows for potentially large bandwidth savings.

- Integration: By utilizing a mobile agent platform designed with a standard object-oriented approach, APHIDS can define clear interfaces that allow new components to be integrated with relative ease. These components can be located anywhere in the network, as a specialized mobile agent can simply migrate to a location suitable to interact with each component.

Chapter 5

Conclusions and Future Work

This thesis documents a mobile agent-based architecture for a programmable IDS which incorporates existing IDS technologies. A detailed description of a prototype implementation, as well as several measurements of performance characteristics of this implementation are provided. A theoretical framework for characterizing the effectiveness of the APHIDS design, and a qualitative analysis of its benefits and disadvantages are also discussed.

To the author's knowledge, this IDS architecture is the first mobile agent-based architecture with a explicit primary goal to use mobile agents as a modular middle-ware layer which ties together separate network and host-based intrusion detection systems. The use of an object-oriented mobile agent platform allows the system to provide useful abstractions for implementing distributed data collection and correlation tasks.

APHIDS incorporates the scalability of mobile agent-based approaches, and also introduces new flexibility, modularity, and real-time reactive analysis features. The APHIDS modular design can incorporate both *misuse* and *anomaly* approaches by implementing components that employ each technique.

The implementation described in this thesis represents an important initial step in exploring the use of mobile agent technology in intrusion detection and analysis. The APHIDS system provides a platform for future enhancement, experimentation, and adaptation. Several future directions of this

work have been identified and are described below:

- A potentially valuable use of mobile agents is to implement a dynamic monitoring and data capture system, which would be able to selectively capture live data related to specific events. For example, based on a detected attack from a particular host, the system could capture all network traffic, as well as traces of all processes started by users from that host. The only way to perform this type of capture currently is to capture all possible data, quickly leading to an explosion in storage requirements. The dynamic and intelligent nature of mobile agents could be exploited to create a system capable of dynamically filtering and recording data on the fly.

- The current APHIDS prototype uses a very simple trigger agent design. Detection engine designs could be borrowed from the literature to develop a modular, flexible trigger agent that is capable of integrating several intrusion detection systems simultaneously. This agent could employ a finite state machine-based design to detect patterns of alerts from existing IDS's.

- The current prototype is geared towards integrating *misuse* type detection engines as input. Integration of *anomaly*-based systems would likely be of value, but this may require an adjustment of the system design.

- When a single event is detected multiple times, the correlation agent will instantiate new copies of task agents each time to perform the distributed search tasks, requiring a full agent to be transferred every time. A caching mechanism could be used to significantly reduce the volume of agents which need to be transferred.

- As discussed in Section 4.3.1, the one-to-one mapping between correlation agents and routines allow for interesting approaches to load balancing and distribution of the processing requirements of the correlation routines. A distributed load balancing algorithm built into

the correlation agent could achieve an effective automatic distribution without prohibitively increasing the correlation agent's size. In general, the scalability of the APHIDS design must be further developed and tested. This may also involve a use of alternative mobile agent platforms.

- Techniques such as semi-random migration of correlation agents and agent replication can be incorporated to improve the system's fault tolerance by rendering correlation agents less susceptible to attack. In general, a thorough investigation into possible measures to protect the APHIDS system itself is required.

- Correlation routines, task agents, and trigger agents need to be developed to handle a greater variety of realistic attack scenarios. This process will not only identify inflexibilities in the APHIDS design but allow for better testing and evaluation procedures. After developing such supporting elements, APHIDS can then be deployed in a real-world scenario and can produce valuable knowledge and experience.

Appendix A

APHIDS Extension API

A.1 CorrelationObject API

To implement new correlation routines, a developer needs to subclass the `aphids.CorrelationObject` class. This class has two abstract methods that need to be overridden:

```
public void doCorrelation();
public AgentInfo deployTrigger(CorrelationAgent ca);
```

The first method should implement the actual correlation routine. The implementation of this method can access any standard Java library, and can use all of Java's available processing facilities. The CorrelationObject superclass implements several utility methods to request the correlation agent to create new task agents or action agents.

```
public void runTask(String agentClassname, String DataID, Map args);
public void runAction(String agentClassname, Map args);
```

The only difference between the semantics of these two methods is that the runTask() function requires a string argument to identify the return value of the task agent's search routine. This value is used for synchronization

purposes: namely, when the correlation object must wait for a task agent to return before continuing processing, it can call the following function:

```
public void waitForResponse(String id);
```

To block the thread of execution until a value has been returned from a task agent which was initialized with DataId == id. As action agents are not intended to return values to correlation routines, their invocation does not require such an id value.

The second method, deployTrigger(), provides a mechanism for the CorrelationObject to define which trigger agent is deployed. This method is responsible for the actual deployment of the trigger, and must return a Grasshopper AgentInfo object so that the correlation agent can gain access to the trigger agent. A reference to the correlation agent is also passed in to enable access to the underlying agent system.

A.2 TriggerAgent API

A new trigger agent is created by subclassing the provided Aphids.TriggerAgent abstract class. The implementor must provide two methods:

```
public void initTrigger(Object args[]);
public void doTrigger();
```

This first method implements initialization logic, with parameters that are passed to it from the a Grasshopper agent's standard init() call. The latter method implements the monitoring logic. When a new trigger event needs to be reported, the following utility function is provided:

```
public void reportTrigger(Map values);
```

This method takes the provided Map object (usually a mapping of String objects to trigger event-specific values) and forwards it to the correlation agent which created this trigger agent.

A.3 TaskAgent API

A new task agent is defined in manner that is very similar to the trigger agent. A new task agent must subclass the provided Aphids.TaskAgent class, and implement the following method:

```
public void doTask();
```

This method implements the task logic for the task agent, and can include calls to the move() function to perform any necessary migration. Any non-transient member variables are automatically serialized and made available after the migration. After the migration, the doTask() method is automatically called again, to allow the agent to continue appropriate processing after migration.

Initialization arguments from the task agent's creating correlation agent are made available with the following function:

```
protected Object getArg(String name);
```

This function directly accesses a copy of the Map object that was provided as arguments to the task agent's creation. Finally, a function is provided for the task agent to communicate its result back to the correlation agent:

```
public void returnValue(Map values);
```

This function is used in a manner exactly similar to the reportTrigger() method of the Trigger Agent. The Map object argument contains String to value mappings of return values that are defined by the task agent.

Appendix B

Visualization

This appendix details related research work on the application of information visualization to the analysis of dynamic mobile agent systems.[1]

The conventional approach in helping a developer to form an understanding of the behavior of a complex distributed system is to obtain a log of relevant system events. This simple approach is often inadequate for the mobile agent scenario for the following reasons:

- *Parallel Events*: A log file typically provides a serial presentation of events. This forces a developer who reads such a file to manually reconstruct those events which occur in parallel. Although manageable in small cases, in a large scenario this task can be time consuming, error-prone, inefficient, and ineffective.

- *Complex Network Topology*: A developer of a mobile agent-based system must not only understand the behavior of the agents, but must also understand the network within which they operate. If this network has a non-trivial topology, this forces to the developer to maintain some sort of mental model of this topology.

Advances in computer graphics algorithms and hardware rendering sys-

[1] A version of this appendix has been accepted for publication. Ken Deeter, Son Vuong, "AgentViz: A Visualization System for Mobile Agents", 1st International Workshop on Mobility Aware Technologies and Applications (MATA 2004), 2004 [8]

tems have produced inexpensive, commonly-available solutions that are capable of rendering and animating many on-screen elements rapidly and smoothly. The ability of these graphics solutions to rapidly display a large number of visual objects has great potential for aiding the networking community. As other works have shown, visual displays can greatly aid the understanding and design of distributed systems.

The remainder of this appendix describes AgentViz, a tool that aims to make use of this rendering technology and apply techniques borrowed from the information visualization field to create a tool to help developers employing the mobile agent paradigm to better understand the behavior of their systems. It provides an offline visualization of a network topology and an animation mobile agents moving within this topology.

B.1 Related Work

To the author's knowledge, there are no published reports directly regarding the visualization of mobile agents. However, visualization of various aspects networking technologies has been explored in the literature.

The work that is closest in spirit is the NAM network animator [11], a visualization system for the NS network simulator. NAM reads specially formatted network trace files and displays and animates packet transmissions to enable viewers to visually comprehend complex temporal network behavior. AgentViz can be viewed as an extension of the NAM idea, specifically targeted toward mobile agent applications.

The distortion techniques used in AgentViz have been thoroughly elaborated upon in the information visualization community[26]. The specific application of distortion to graph drawing has also been explored [35].

Although AgentViz avoids the *graph layout* problem by requiring network nodes to specify positional coordinates, it is nonetheless a relevant problem. The problem has been explored in several approaches, including the Otter [20] and Walrus systems from CAIDA, the DynaDAG system [32], as well as the Gnutellavision system [43]. Each approach makes different assumptions about the nature of the topology being visualized and the interaction that

is needed between the user and the system, leading to differing layout and rendering strategies.

The usefulness of visualization as a illustration and communication tool has been explored as well [41, 19]. AgentViz can also be used as an illustrative and educational tool, much in the same manner as the systems described in these reports.

B.2 Design Goals

The main intention for AgentViz is to be used as a debugging and verification tool. It can help a developer to understand and confirm that a system is working according to design. We assume that the user has a strong comprehension of the system they are wishing to visualize, and thus the goal is less to provide a tool that allows a viewer to make inferences about a system they do not know, but rather to provide a tool with which a user can match a pre-existing mental model to a visual one.

A major goal in the design of AgentViz was to architect it in a manner that allows it to be leveraged in as many different situations as possible. In general, a visualization tool should not dictate the design of a system for which it is meant to visualize. In designing AgentViz, this principle was followed to the extent possible.

As is the case with many other visualization systems, the key benefit is the reduction of "cognitive load" on the part of the user. Specifically, this involves reducing or removing the need for a user to maintain a mental model of a network topology, to remember absolute and relative spatial orientations in this model, or to reconstruct parallel sequences of events. By aiding in these tasks, a user is able to devote more cognitive resources to the direct goal of understanding a complex dynamic system.

AgentViz also aims to employ techniques developed in the information visualization field where they are relevant and effective. Distortion and zooming are used to allow the exploration of network topology, and *visual encoding* techniques are used to display abstract simulation state data in a user-configurable way.

Entity Type	Visual Representation	Visual Parameters
Node	Circle	Color, Size
Link	Line	Color, Thickness
Agent	Square	Color, Size

Table B.1: Three different entity types are available in the AgentViz network model. The table lists each type, along with their visual representations and parameters.

B.3 AgentViz

B.3.1 Network Model

AgentViz maintains an internal network model that is structured in a way to allow efficient rendering. This model, while basic, provides a fundamental set of *entities* to simulate a mobile agents operating over a network.

The model supports three *entity types*: Nodes, Links, and Agents. Each entity type has several visual parameters. Each entity type has a different set of parameters, corresponding to their differing visual representations. The visual representation and visual parameters for each entity type is summarized in Table B.1.

Each node entity also records a position as a two-dimensional coordinate in a space defined by the simulation (this space is referred to as *model space*). The positions of Links and Agents are determined by the nodes that are related to them. In the case of links, their position is simply defined by the positions of their two endpoint nodes. For agents, the position is determined by either the position of the node at which the agent currently resides, or somewhere along a link if the agent is in transit.

The model also keeps track of a set of *entity states* for each entity type. An entity state is simply a set of values for the visual parameters of an entity type. For example, a *node state* would specify one particular combination of a color and a size measure. [2]

The real-world meaning of each visual state is arbitrary, and is meant

[2]For nodes, the size corresponds to the radius of the node

to determined by users of the system. Whereas one system might use a "large red node" to encode a significant node in a P2P overlay network, another system might use the same visual appearance to differentiate routers from ordinary hosts. The choice to leave the *visual encoding* configurable was made to meet the goal of generic design. By allowing the data source to manage the visual encoding, the tool can be adapted to display many different types of states for many different systems.

Time in the model is represented using a generic unit referred to as a *tick*. The ratio of ticks to seconds is determined by the simulation producing the input to AgentViz. The tick to time ratio can also be adjusted in the user interface to allow "slow motion" or "fast forward"-like time distortion effects.

B.3.2 Network Events

The input to the AgentViz tool is simply a file containing a listing of *network events*. This file is formatted using a simple syntax defined by AgentViz, and any system that is capable of generating a file in this format can be visualized using this tool.

Each *event* has a type, which encodes its meaning in the context of the network model. Along with the type are specified arguments, which are specific to the event's type. The types of events available are summarized in Table B.2.

Abstractly, each *event* is considered an operation upon the network model. When a log file is read, the initial network model is initialized to be empty. Events that signify the creation of various network entities are used to create the initial topology of the network. As the system proceeds through the events, new network entities can be created and destroyed by the corresponding events. This mechanism allows the system to support the visualization of non-static topologies – a common scenario in many mobile agent systems.

Along with each event, two timestamps are specified. The first records the time at which an event began, and one for the time at which it ended. By

Event Type	Event Effect
`create_node`	Creates a new node
`create_link`	Creates a new link
`create_agent`	Creates a new agent
`destroy_node`	Removes specified node from the model
`destroy_link`	Removes specified link from the model
`destroy_agent`	Removes specified agent from the model
`node_state_change`	Changes state of specified node
`link_state_change`	Changes state of specified link
`agent_state_change`	Changes state of specified agent
`agent_move*`	Indicates agent movement
`node_move*`	Indicates node movement

Table B.2: Listed are the types of events that can be recorded in the log file provided as input to AgentViz, along with their effects on the network model. Events with an asterisk (*) indicate those which have two timestamps to indicate when they begin and end

having two timestamps, the system is able to reconstruct events that occur in parallel, and that take a period of time to complete. In the current design, only the agent_move and node_move events apply. Most of the supported events are instantaneous (a creation of a node, for example), in which case only the first of the two timestamps is considered during processing.

Along with the events for the creation and destruction of network entities, a third set records state changes of these entities. A state change event tells the system to associate a particular network entity with a new visual state.

A log file can often be incorrect or inconsistent. For example, a `destroy_node` event may call for the destruction of a node which does not exist. Although AgentViz could have been designed to be resilient to these kinds of inconsistencies, the opposite approach is taken. The consistency of the network model is left entirely to the system generating the log file. We have found that, in practice, inconsistencies in the log file are often indicators of bugs or incorrect design in the simulation (or incorrect implementation of a logging mechanism). Therefore, attempting to hide or tolerate these inconsistencies can result in real bugs becoming less visible.

Header Type	Description
`speed`	The default ticks/second for playback
`fixed_bounding_box`	A fixed are of the model space to display
`define_link_state`	Defines a visual state for a link
`define_node_state`	Defines a visual state for a node
`define_agent_state`	Defines a visual state for an agent
`render_link_priority`	Order in which links should be rendered
`render_node_priority`	Order in which nodes should be rendered
`render_agent_priority`	Order in which agents should be rendered

Table B.3: A listing of the available log header types.

A log file can also contain headers, which are used to specify parameters for the visualization that remain static as the log is played back. The list of available header types is summarized in Table B.3.

The most common usage of headers are to define entity visual states. Each entity state is associated to a string identifier, which can be used elsewhere in the file to refer to that state. Events that change the states of entities must refer to these states by name, and cannot create new states on the fly.

B.3.3 Rendering Pipeline

The rendering system in AgentViz is architected as a pipeline with four stages: Layout, Zoom, Distort, and Render. A high-level view of this pipeline is provided in Figure B.1. The OpenGL API used by AgentViz is a stateful library, and causing unnecessary state changes within the library can lead to decreased performance. The pipeline design allows not only for a clear definition of the rendering procedure, but allows the implementation to rapidly perform similar functions on the data set without requiring costly conditional branches and state changes during the rendering process. Also, utilizing abstract interfaces between each stage provides for the decoupling of their implementations, allowing for experimentation by modifying each stage individually.

In the *layout* stage, the positions in model space for each node are determined. Currently, this stage simply uses the positions provided in the input log file. However, this stage exists as a distinct stage to allow for automatic layout mechanisms to be added later.

In the *zoom* stage, model space coordinates are translated to screen-space coordinates. The translation can be performed in a variety of ways. One available method calculates a bounding box around all the nodes in the model, then maps the corners of this box to the corners of the available display window. Another method uses the same process except that the bounding box is calculated for a specific subset of the nodes, based on a user's "selection". Alternatively, if the simulation wishes to dictate this mapping mechanism, it can provide a fixed bounding box using a log header.

In the *distort* stage, the screen-space coordinates of each node is modified by a distortion algorithm which creates virtual "magnification area" whose center can be specified using the mouse. The distance from each node to this center is scaled inverse proportionately to the distance from the magnification center. Nodes that are close the center are spread apart, and nodes that are far away are pushed together, creating a local magnification effect.

Finally, in the *render* stage, the screen-space coordinate of each node is used to draw all entities onto screen. Links are rendered first, followed by nodes, then agents. Within each entity type, all entities with the same state are rendered together. The order in which each state is rendered can either be specified in the log file or can be modified using the user interface. This mechanism allows certain entity types to be rendered on top of others, allowing for clarification and emphasis in dense displays. This also reduces the number of calls made to the OpenGL library to indicate a new color to be used for rendering.

B.3.4 Implementation Details

AgentViz is implemented using Java Standard Edition 1.4 and the Swing GUI toolkit. The rendering component uses a freely available Java binding binding for the OpenGL graphics library, known as jogl.

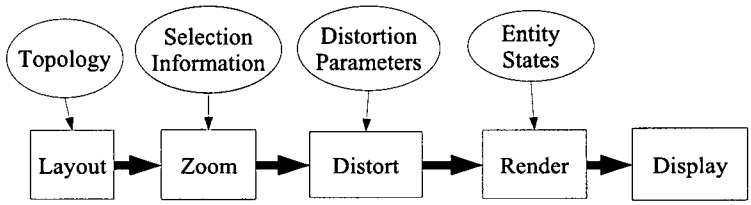

Figure B.1: This figure shows a graphical depiction of the AgentViz's rendering pipeline. Each rendering pass proceeds through the Layout, Zoom, Distort, and Render stages, and the result is sent to the display. Each oval represents the information that can affect the behavior at each stage of the process

B.4 Applications

This section describes several usage scenarios of this tool. The first was developed specifically to illustrate the usefulness of the visualization, while the latter was developed entirely independently.

As previously mentioned, AgentViz can be used as both a debugging and verification tool, as well as an illustrative tool. Summarized below is a general methodology for using the tool from an application programmers point of view.

1. Develop Visual Mappings: as AgentViz does not include any predefined mappings for application state to visual state, a developer must first decide which states will be mapped to which visual parameters.

2. Modify Application: Once a visual mapping has been decided, the target application for visualization needs to be modified to output an event log.

3. Visualize and Verify: The event log is fed into AgentViz, and played back.

Developing a visual mapping is an iterative process, and thus several cycles of the steps above will likely be performed. AgentViz could have provided

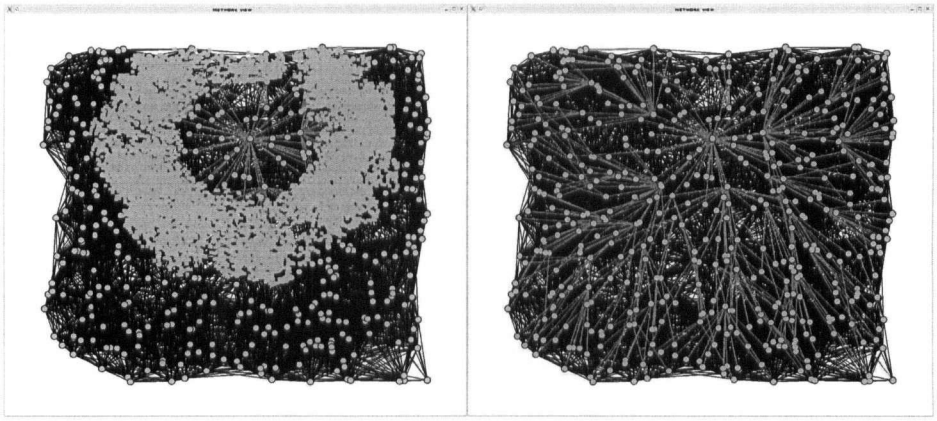

Figure B.2: Two screenshots of a multicast tree creation algorithm in progress over a dense network of 500 nodes. Green squares show the location of mobile agents as they spread recursively from the root node. Red links show links that have been picked to form the multicast tree. The left image shows the algorithm in process and the right image shows the finished result. Black links and red links are represented in the log file as different link states, and the rendering priority for links is configured to render red links in front of black ones, to emphasize the overlay multicast tree

some common visual mapping presets, but this would have required more complexity in the event log format to express more detailed state changes, and could have sacrificed the system's value as a generic tool.

B.4.1 Multicast tree creation using mobile agents

A small simulation was developed to generate a log file for a mobile agent-based multicast tree creation algorithm. The simulation was created using a small python program (~200 lines). The program simulated, a group of agents forms a multicast tree by beginning at a root node and replicating and spreading recursively, looking for shortest paths to each node from the root. Figure B.2 shows two snapshot of the animation of this process.

This example shows several advantages of the tool. First, because the visualization system has been separated from the running instance of the

application, the performance characteristics of each component are also independent. The python-simulation produces output in 'virtual' time, making simple assumptions such as hard coding the time for an agent to transfer as 10 ticks. Even if the simulation could not run in real time, as long as the time units are chosen correctly, the visualization can present a realistic case.

Secondly, this example illustrates that a small program can be used to produce a fairly involved graphical output using this tool. The details of rendering performance have been taken care of, and the application developer can concentrate on correct implementation. The efficiency of development can enable a user to quickly explore new agent interactions and system structure.

B.4.2 Scatternet formation using mobile agents

Our second example shows a visualization of a Bluetooth scatternet formation algorithm based on mobile agents[16]. The algorithm uses mobile agents to traverse existing scatternets to find appropriate nodes that can accommodate a new host that wishes to join the scatternet. A screen shot of AgentViz being used to visualize this process is shown in Figure B.3.

B.5 Conclusions and Future Work

AgentViz is a visualization tool to aid developers of mobile agent-based systems to understand the parallel behavior of their mobile agent systems operating over large network topologies. The software is designed in a generic way allowing it to be adapted to different mobile agent scenarios.

The current implementation provides the most basic features. Many improvements can be incorporated, including:

- Dynamic Layout: Artificial topology generators often do not output positional coordinates for network nodes. To use these topologies in the current system, the user would have to manually derive coordinates for a potentially large number of nodes. An automatic layout mechanism

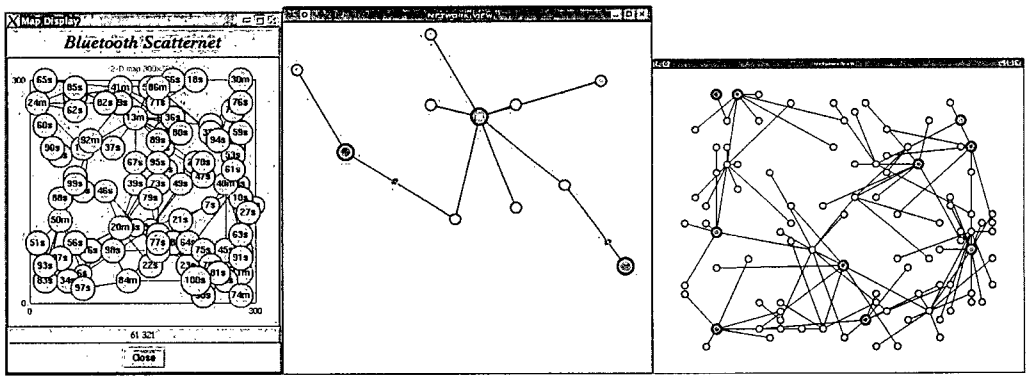

Figure B.3: Left: a screen shot from the visualization that is part of the original simulation package on which this algorithm was developed. Middle: a view of AgentViz animating the mobile agents involved in the scatternet creation. Right: image shows AgentViz displaying the finished scatternet. In the AgentViz images, Bluetooth "master" devices are shown as larger red nodes.

would allow these types of networks to be visualized without incurring a large burden on the user.

- More entities and visual encodings: AgentViz's network model can be extended to support a richer set of network entities, to allow the model to express a larger range of situations. The same goal can be achieved also by increasing the dimensions for visual encodings (allowing differentiation of states using shape or texture, for example). We would also like to explore mechanisms to more directly model the characteristics of wireless hosts, as the application of mobile agents in wireless scenarios is becoming increasingly significant.

- Distortion and Exploration techniques: Although the fisheye-like distortion mechanism in AgentViz provides a useful tool to browse local areas of the network, further distortion techniques can be integrated to allow for new and useful types of interaction.

Bibliography

[1] Spyros Antonatos, Kostas G. Anagnostakis, and Evangelos P. Markatos, *Generating realistic workloads for network intrusion detection systems*, Proceedings of the fourth international workshop on Software and performance, ACM Press, 2004, pp. 207–215.

[2] Midori Asaka, Atsushi Taguchi, and Shigeki Goto, *The implementation of ida: An intrusion detection agent system*, Proceedings of the 11th FIRST Conference, 1999.

[3] C. Baumer, M Breugst, M Choy, and T Magedanz, *Grasshopper - a universial agent platform based on omg masif and fipa standards*, First International Workshop on Mobile Agents for Telecommunication Applications (MATA '99), 10 1999, pp. 1–18.

[4] Suresh N. Chari and Pau-Chen Cheng, *Bluebox: A policy-driven, host-based intrusion detection system*, ACM Trans. Inf. Syst. Secur. **6** (2003), no. 2, 173–200.

[5] Mark Crosbie and Eugene H. Spafford, *Applying genetic programming to intrusion detection*, Working Notes for the AAAI Symposium on Genetic Programming (MIT, Cambridge, MA, USA) (E. V. Siegel and J. R. Koza, eds.), AAAI, 10–12 1995, pp. 1–8.

[6] Mark Crosbie and Gene Spafford, *Defending a computer system using autonomous agents*, 8th National Information Systems Security Conference, 1996.

[7] Ken Deeter, Son Vuong, Kapil Singh, Steve Wilson, and Luca Filipozzi, *Aphids: A mobile agent-based programmable hybrid intrusion detection system*, 1st International Workshop on Mobility Aware Technologies and Applications (MATA 2004), 2004.

[8] Ken Deeter and Son T. Vuong, *Agentviz: A visualization system for mobile agents*, 1st International Workshop on Mobility Aware Technologies and Applications (MATA 2004), 2004.

[9] Jose Duarte de Queiroz, Luiz Fernando Rust da Costa Carmo, and Luci Pirmez, *Micael: An autonomous mobile agent system to protect new generation networked applications*, 2nd Annual Workshop on Recent Advances in Intrusion Detection, 1999.

[10] Robert Durst, Terrence Champion, Brian Witten, Eric Miller, and Luigi Spagnuolo, *Testing and evaluating computer intrusion detection systems*, Commun. ACM **42** (1999), no. 7, 53–61.

[11] Deborah Estrin, Mark Handley, John Heidemann, Steven McCanne, Ya Xu, and Haobo Yu, *Network visualization with nam, the vint network animator*, Computer **33** (2000), no. 11, 63–68.

[12] Noria Faukia, David Billard, and Juergen Harms, *Computer system immunity using mobile agents*, HP Openview University Association 8th Annual Workshop, 2001.

[13] Noria Faukia, Salima Hassas, Serge Fenet, and Paul Albequerque, *Combining immune system and social insect metaphors: A paradimg for intrusion detection and response system*, Proceedings of the 5th International Workshop for Mobile Agents for Telecommunication Applications, 2003.

[14] Jeremy Frank, *Artificial intelligence and intrusion detection: Current and future directions*, Proceedings of the 17th National Computer Security Conference (Baltimore, MD), 1994.

[15] Alfonso Fuggetta, Gian Pietro Picco, and Giovanni Vigna, *Understanding code mobility*, IEEE Transactions on Software Engineering **24** (1998), no. 5, 342–361.

[16] Sergio Gonzalez-Valenzuela, Sont T. Vuong, and Victor C. M. Leung, *Efficient formation of dynamic bluetooth scatternet via mobile agent processing*, Proceedings of 5th Iternational Workshop for Mobile Agents for Telecommunication Applications, 2003.

[17] Robert S. Gray, *Agent Tcl: A transportable agent system*, Proceedings of the CIKM Workshop on Intelligent Information Agents, Fourth International Conference on Information and Knowledge Management (CIKM 95) (Baltimore, Maryland), December 1995.

[18] Guy Helmer, Johnny S. K. Wong, Vasant Honavar, Les Miller, and Yanxin Wang, *Lightweight agents for intrusion detection*, Journal of Systems and Software **67** (2003), no. 2, 109–122.

[19] Mark A. Holliday, *Animation of computer networking concepts*, ACM Journal of Educational Resources in Computing **3** (2003), no. 2, 1–26.

[20] Bradley Huffaker, Evi Nemeth, and K Claffy, *Otter: A general-purpose network visualization tool*, Proceedings of ISOC INET'99, 1999.

[21] Koral Ilgun, Richard A. Kemmerer, and Phillip A. Porras, *State transition analysis: A rule-based intrusion detection approach*, Software Engineering **21** (1995), no. 3, 181–199.

[22] M. ITA, *Concordia: An infrastructure for collaborating mobile agents.*

[23] J O Garcia Fernandez J S Balasubramaniyan, D Isacoff, E Spafford, and D Zamboni, *An architecture for intrusion detection using autonomous agents*, Tech. Report 98/05, COAST Laboratory, Purdue University, 1998.

[24] Christopher Kruegel and Thomas Toth, *Sparta - a mobile agent based intrusion detection system*, Proceedings of the IFIP Conference on Network Security (I-NetSec), 2001.

[25] D. Lange, M. Oshima, G. Karjoth, and K. Kosaka, *Aglets: Programmable mobile agent in java*, Proceedings of Worldwide Computing and its Applications (WWCA '97), vol. 1247, Lecture Notes in Computer Science, 1997.

[26] Y.K. Leung and M.D. Apperley, *A review and taxonomy of distortion-oriented presentation techniques*, ACM Transactions on Computer-Human Interaction **1** (1994), no. 2, 126–160.

[27] Chunsheng Li, Qingfeng Song, and Chengqi Zhang, *Ma-ids architecture for distributed intrusion detection using mobile agents*, Proceedings of the 2nd International Conference on Information Technology for Application (ICITA 2004), 2004.

[28] Richard Lippmann, David Fried, Isaac Graf, Joshua Haines, Kristopher Kendall, David McClung, Dan Weber, Seth Webster, Dan Wyschogrod, Robert Cunningham, and Marc Zissman, *Evaluating intrusion detection systems: The 1998 DARPA off-line intrusion detection evaluation*, Proceedings of the DARPA Information Survivability Conference and Exposition (Los Alamitos, CA), IEEE Computer Society Press, 2000.

[29] Roy A. Maxion and Kymie M. C. Tan, *Benchmarking anomaly-based detection systems*, Proceedings of the 2000 International Conference on Dependable Systems and Networks (formerly FTCS-30 and DCCA-8), IEEE Computer Society, 2000, pp. 623–630.

[30] John McHugh, *Testing intrusion detection systems: a critique of the 1998 and 1999 darpa intrusion detection system evaluations as performed by lincoln laboratory*, ACM Trans. Inf. Syst. Secur. **3** (2000), no. 4, 262–294.

[31] Dejan S. Milojicic, Markus Breugst, Ingo Busse, John Campbell, Stefan Covaci, Barry Friedman, Kazuya Kosaka, Danny B. Lange, Kouichi Ono, Mitsuru Oshima, Cynthia Tham, Sankar Virdhagriswaran, and Jim White, *Masif: The omg mobile agent system interoperability facility*,

Proceedings of the Second International Workshop on Mobile Agents, Springer-Verlag, 1999, pp. 50–67.

[32] Stephen C North, *Incremental layout in dynadag*, Proceedings of Graph Drawing '95, 1996.

[33] Stefan Poslad and Patricia Charlton, *Standardizing agent interoperability: the fipa approach*, Mutli-agents systems and applications (2001), 98–117.

[34] M Roesch, *Snort – lightweight intrusion detection system for networks*, Proceedings of USENIX LISA'99, 11 1999.

[35] Manojit Sarkar and Marc H. Brown, *Graphical fisheye views of graphs*, Tech. Report 084a, Digital Systems Research Center, 1992.

[36] Robin Sommer and Vern Paxson, *Enhancing byte-level network intrusion detection signatures with context*, Proceedings of the 10th ACM conference on Computer and communication security, ACM Press, 2003, pp. 262–271.

[37] Hock Kim Tan and Luc Moreau, *Certificates for mobile code security*, Proceedings of the 2002 ACM symposium on Applied computing, ACM Press, 2002, pp. 76–81.

[38] G. Vigna, B. Cassell, and D. Fayram, *An intrusion detection system for aglets*, Proceedings of the International Conference on Mobile Agents (Barcelona, Spain) (N. Suri, ed.), LNCS, Springer, 10 2002.

[39] G. Vigna and R. Kemmerer, *NetSTAT: A Network-based Intrusion Detection Approach*, Proceedings of the 14^{th} Annual Computer Security Application Conference (Scottsdale, Arizona), December 1998.

[40] Giovanni Vigna, *Cryptographic traces for mobile agents*, Lecture Notes in Computer Science **1419** (1998), 137–??

[41] Curt M. White, *Visualization tools to support data communications and computer network courses*, Journal of Computing in Small Colleges **17** (2001), no. 1, 81–89.

[42] Yu-Sung Wu, Bingrui Foo, Yongguo Mei, and Saurabh Bagchi, *Collaborative intrusion detection system (cids): A framework for accurate and efficient ids*, Proceedings of the 19th Annual Computer Security Applications Conference (ACSAC'03), 2003.

[43] Ka-Ping Yee, Danyel Fisher, Rachna Dhamija, and Marti A. Hearst, *Animated exploration of dynamic graphs with radial layout*, Proceedings of INFOVIS, 2001, pp. 43–50.