APPLYING MOBILE AGENTS IN AN IMMUNE-SYSTEM-BASED INTRUSION DETECTION SYSTEM

by

MAREK PIOTR ZIELINSKI

Submitted in part fulfilment of the requirements
for the degree of

MASTER OF SCIENCE

in the subject

COMPUTER SCIENCE

at the

UNIVERSITY OF SOUTH AFRICA

SUPERVISOR: PROF LM VENTER

NOVEMBER 2004

ACKNOWLEDGEMENT

This research has been supported by a grant from the National Research Foundation. The opinions, findings, and conclusions expressed in this material are that of the author and are not necessarily to be attributed to the National Research Foundation.

ABSTRACT

APPLYING MOBILE AGENTS IN AN IMMUNE-SYSTEM-BASED INTRUSION DETECTION SYSTEM

Nearly all present-day commercial intrusion detection systems are based on a hierarchical architecture. In such an architecture, the root node is responsible for detecting intrusions and for issuing responses. However, an intrusion detection system (IDS) based on a hierarchical architecture has many single points of failure. For example, by disabling the root node, the intrusion-detection function of the IDS will also be disabled.

To solve this problem, an IDS inspired by the human immune system is proposed. The proposed IDS has no single component that is responsible for detecting intrusions. Instead, the intrusion-detection function is divided and placed within mobile agents. Mobile agents act similarly to white blood cells of the human immune system and travel from host to host in the network to detect intrusions. The IDS is fault-tolerant because it can continue to detect intrusions even when most of its components have been disabled.

Key terms:

Computer security; intrusion detection system; immune system; mobile agent; fault-tolerance; anomaly detection; system call monitoring; computer immunology.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABBREVIATIONS USED

ACL	Agent Communication Language
API	Application Programming Interface
CIDF	Common Intrusion Detection Framework
CORBA	Common Object Request Broker Architecture
CPU	Central Processing Unit
I/O	Input / Output
IDS	Intrusion Detection System
IP	Internet Protocol
LFC	Locality Frame Count
RMI	Remote Method Invocation
RPC	Remote Procedure Call

CHAPTER 1

INTRODUCTION

1.1 Intrusion detection systems

A computer system's security mechanisms should prevent unauthorized access to its resources and data. However, it is impossible to build a completely secure system for many different reasons: programs and operating systems have vulnerabilities, firewalls can be circumvented, passwords can be cracked, and a system can be abused by insiders (Sundaram, 1996). Moreover, the increasing connectivity of computer systems gives greater access to outsiders and makes it easier for intruders to avoid identification (Mukherjee, Heberlein & Levitt, 1994).

Unless there is a mechanism that will detect breaches of the system's security, we may be unaware that the computer system has been attacked. *Intrusion detection* provides such a mechanism and is defined as "the problem of identifying individuals who are using a computer system without authorization (i.e. 'crackers') and those who have legitimate access to the system but are abusing their privileges (i.e. the 'insider threat')" (Mukherjee, Heberlein & Levitt, 1994). A computer system that provides this function is called an Intrusion Detection System (IDS).

1.2 Problem statement

Nearly all present-day commercial intrusion detection systems (IDSs) follow a hierarchical architecture (Jansen, 2002; Jansen et al. 2000; Jansen et al. 1999). Hierarchical architectures are used because they are excellent for creating scalable distributed IDSs with central points of administration. However, an IDS based on a hierarchical architecture has many single points of failure if it has no redundant communication lines or the capability to dynamically reconfigure relationships in the

case of failure of key components. For example, an attacker may interrupt the operation of the entire IDS by successfully disabling the system's root node. Since the root node is responsible for the actual function of detecting intrusions and for issuing responses, disabling it will not allow the IDS to detect intrusions (this is further explained in Section 2.5.2). Therefore, a hierarchical architecture does not facilitate the building of robust and fault-tolerant IDSs.

An intruder who knows that an IDS is used may want to disable it, thereby allowing the intruder to attack the rest of the computer system undetected. The critical role played by the root node of the IDS makes it a likely target of attack. Although such critical components usually reside on platforms that have been hardened to resist direct attack, the IDS may still be vulnerable as other survivability techniques such as redundancy, mobility, or dynamic recovery are lacking in current implementations. A system could also employ redundant components for each key node to avoid this problem. However, such a solution does not offer much fault-tolerance because a determined and knowledgeable attacker can disable a small number of backups.

From the above discussion, it is clear that there exists a need to improve the described hierarchical architecture in order to create a more robust and fault-tolerant IDS. Therefore, in this dissertation, the following question will be investigated:

> *Can an IDS architecture be proposed such that it will offer greater fault-tolerance and consequently overcome the described limitation (i.e. vulnerability to direct attack) of current hierarchical IDS architectures that nearly all present-day commercial IDSs use?*

1.3 Solution approach

To solve the stated problem, we propose an IDS inspired by the human immune system. The human immune system has many properties that not only allow it to effectively detect and eliminate intrusions (such as bacteria) in the body, but they also provide fault-tolerance in the system. The immune system can remain functional even when many of its components have failed. When certain properties of the human immune

system are applied to intrusion detection, the result is an IDS that also provides fault-tolerance.

As in the immune system, the proposed IDS uses small, independent, and disposable intrusion detectors. The role of the intrusion detectors is played by mobile agents, which travel from host to host in the network to detect possible intrusions. When mobile agents are applied in an IDS based on the immune system, the result is an IDS that is made more resistant to failure because it can remain operational even when most of its components have been disabled.

Several approaches for building computer security architectures that incorporate principles of the human immune system have been proposed by Somayaji, Hofmeyr and Forrest (1997). One approach suggested was to implement the adaptive immune system layer by kernel-assisted lymphocyte processes, or mobile agents, that can migrate between computers. With help from the kernel, the lymphocyte processes are able to query other processes to determine if they are functioning normally. Nevertheless, the authors did not provide further details about this particular suggested approach. We used this suggestion in our earlier work (Zielinski & Venter, 2004) and proposed an IDS based on certain properties of the human immune system. In this dissertation, the proposed IDS is discussed in much greater detail.

1.4 Scope of this research

The discussion of the proposed IDS is limited to only a theoretical description of the system in terms of its main components and how it should function if it were implemented. In addition, the proposed IDS is described only in terms of its intrusion-detection function. The intrusion-response function of the IDS is not considered in this work.

This dissertation also does not discuss aspects of the proposed IDS that depend on a particular operating system under which the IDS will run, or which depend on how the IDS is implemented. The motivation for this choice is that, by not considering implementation issues when proposing the IDS, the applicability of the proposed IDS is

not restricted to only certain environments. That is, it also allows the implementers to address implementation issues in a way that is optimal for a particular environment, rather than being restricted to using aspects of the system that are optimal for only a certain environment. This dissertation does, however, discuss some of the most significant issues that should be considered when implementing the proposed IDS.

Furthermore, because this research is a conceptual study of how mobile agents can be applied in an immune-system-based IDS, the proposed IDS has not been tested by, for example, creating a prototype of the system.

1.5 Organization of chapters

This dissertation is organized as follows:

- Chapter 1 is a brief introduction.

- Chapter 2 provides a discussion of intrusion detection systems. Topics discussed include: the need for IDSs, how IDSs are classified, different approaches to intrusion detection, different IDS architectures, desirable properties of IDSs, as well as the current shortcomings of IDSs.

- Chapter 3 provides background information to mobile agents. Topics discussed include: the components of a mobile agent system, agent behaviour, advantages and disadvantages of applying mobile agents to intrusion detection, as well as security issues of mobile agents.

- Chapter 4 briefly discusses the human immune system and the properties that enable it to effectively detect intruders (such as bacteria) in the body.

- Chapter 5 discusses the proposed IDS. This chapter also contains a discussion of how concepts of the human immune system have been applied in the proposed IDS and how the proposed IDS is similar to the human immune system.

- Chapter 6 evaluates the proposed IDS in terms of how it solves the research problem, as well as the system's benefits and drawbacks. Some of the most significant issues that will need to be considered when implementing the IDS are also discussed.

- Chapter 7 contains concluding remarks as well as recommendations for further research.

CHAPTER 2

INTRUSION DETECTION SYSTEMS

2.1 Introduction

An intrusion is an event, or a set of events, that attempts to compromise a computer system's confidentiality, integrity, availability, or that attempts to bypass its security mechanisms. Intrusions can be caused by system insiders or by external attackers. System insiders, or users authorized to use the system, can cause intrusions by attempting to gain privileges to which they are not entitled or by misusing the privileges that have been given to them. External attackers, or users who have not been authorized to use the system, can cause intrusions by gaining access to the system from outside, such as the internet (Bace & Mell, 2001).

An intrusion detection system (IDS) is an automated system that aims to detect intrusions in a computer system. The main goal of an IDS is to detect any unauthorized use, abuse, or misuse of computer systems by both system insiders and external attackers (Mukherjee, Heberlein & Levitt, 1994). Its purpose can be compared to that of a car alarm, which alerts its owner when the car has been broken into. Once an intrusion has been detected, the IDS typically issues an intrusion-response action, which may range from reporting the intrusion to the system administrator, to taking some action against the intruder.

In this chapter, a detailed discussion of intrusion detection systems is presented. This chapter is organized as follows. Section 2.2 lists the reasons why intrusion detection systems are necessary. This is followed by Section 2.3, which contains a classification of the different types of IDSs together with their advantages and disadvantages. Section 2.4 discusses the different components of a typical IDS. Organizing the components in different ways results in different IDS architectures, which are discussed in Section 2.5.

This is followed by a discussion of the desirable characteristics of an IDS in Section 2.6 and the current shortcomings of IDSs in Section 2.7. This chapter is concluded in Section 2.8.

2.2 Why are intrusion detection systems necessary?

Increased network connectivity of computer systems gives greater access to outsiders and makes it easier for intruders to avoid identification (Mukherjee, Heberlein & Levitt, 1994). By being connected to the internet, computer systems are exposed to different threats and are made more vulnerable to different attacks. By using an IDS, an attack on the computer system can be detected and measures can be taken to stop it before any damage is done to the computer system.

There are several reasons why IDSs are necessary (Bace & Mell, 2001):

- *To serve as a means to deter those who would violate security policy.* This assumes that an increased perceived risk of discovery and prosecution of attackers can prevent certain security problems. That is, an attacker may be deterred from, for example, gaining unauthorized access to the system due to the possibility of being discovered and prosecuted.

- *To detect attacks and other security violations that other security measures cannot prevent.* An IDS can be used to detect attacks that exploit vulnerabilities in the security mechanisms of a computer system. In addition, an IDS can serve an important function in protecting the system because it can report intrusions to system administrators, who can contain and recover any resulting damage.

- *To detect preambles of attacks.* The first stage of an attack usually involves examining a system or network for any vulnerabilities, searching for an optimal point of entry. This stage is often experienced as network probes and other tests for existing vulnerabilities. By using an IDS, the probes can be detected and action may be taken to block the attacker's access to the target system.

- *To document the existing system threat.* An understanding of the frequency and characteristics of attacks allows understanding of what security measures are appropriate to protect the system.

- *To act as a means of quality control for security design and administration.* An IDS that runs over a period of time can show patterns of system usage and detected problems. These can show the design and management flaws in the system's security. Deficiencies can be corrected before they cause a security problem.

- *To provide information about actual intrusions.* An IDS can collect relevant and detailed information about the attack, which supports incident handling and recovery efforts. Such information can also be used to identify problem areas in the security configuration or policy of the system.

2.3 Classification of intrusion detection systems

Intrusion detection systems can be classified according to:

1. The source of data used for analysis
2. The intrusion detection model used by the IDS
3. The distribution of the IDS components

2.3.1 The source of data used for analysis

An IDS requires specific types of data that it can analyse for possible intrusions. The data is obtained from different sources, depending on what types of attacks should be detected by the IDS. With respect to the source of data used for analysis, intrusion detection systems are classified as host-based or network-based (Mukherjee, Heberlein & Levitt, 1994).

2.3.1.1 Host-based IDSs

A host-based IDS operates at the host level, collecting information from within an individual computer system (Bace & Mell, 2001). This allows a host-based IDS to analyse activities with high precision and reliability; it can determine exactly which processes and users are involved in a particular attack on the operating system. A host-based IDS also has the ability to directly access and monitor the data files and system processes usually targeted by attacks. Therefore, it can view the system after an attempted attack, which allows it to verify the success or failure of an attack.

A host-based IDS normally uses information sources of two types: operating system audit logs and system logs (Bace, 2000). Operating system audit logs are records of system events, generated at the innermost (kernel) level of the operating system. System logs, on the other hand, are files of system and application events.

An audit log can reveal system events at a finer-grained level of detail than system logs. However, this makes audit logs larger and more complex to understand than system logs. System logs are also considered less trustworthy than operating system audit logs. This is because the log-generation program is usually running as an application, which makes it easier to subvert or otherwise modify than the audit subsystem. In addition, because the system logs are usually stored in unprotected directories on the system, it is easy for an attacker to locate and modify or destroy the files in an effort to remove evidence of an attack.

A special subset of host-based IDSs are application-based IDSs, which analyse the events that occur within a software application. An application's transaction log files typically serve as the data source for this subset of host-based IDSs.

Application-based IDSs have the ability to interface with the application directly, with significant domain or application-specific knowledge included in the analysis engine. This allows detection of suspicious behaviour due to authorized users exceeding their authorization.

2.3.1.2 Network-based IDSs

A network-based IDS uses raw network packets as its data source. The network packets are obtained from a network adapter running in promiscuous mode (a mode during which all the frames that pass over the network are picked up; not just those destined for the node served by the card). Early IDSs were host-based, but present systems are usually network-based. Although most network-based IDSs build their detection mechanism on monitored network traffic, some also use host audit logs (Mukherjee, Heberlein & Levitt, 1994).

A network-based IDS often has several single-purpose sensors or hosts placed at various points in a network. These units monitor network traffic by performing local analysis of that traffic and report attacks to a central management console (Bace & Mell, 2001). Since a network-based IDS can monitor the network traffic as an attack is occurring, it is more difficult for an attacker to remove evidence of attacks than in a host-based IDS.

2.3.1.3 Comparison of host- and network-based IDSs

Host- and network-based IDSs have different strengths and weaknesses resulting from the source of data used for analysis and the environment in which they operate (ISS, 1998; Bace & Mell, 2001).

Due to the different sources of data used for analysis, a host-based IDS can detect intrusions that a network-based IDS will miss, and vice versa. For example, an attack from the keyboard of a critical server cannot be seen by a network-based IDS as no data passes the network, but this could be detected by a host-based IDS. On the other hand, many IP-based denial-of-service and fragmented packet attacks can only be detected by examining packet headers as they travel across a network. Since a host-based IDS cannot examine all packet headers, it cannot detect these types of attacks, which can be detected by a network-based IDS.

A network-based IDS monitors network traffic in a way that is transparent to the users of the network. Therefore, the likelihood than an attacker will be able to locate the

network monitor and disable it without significant effort is decreased. In contrast, a host-based IDS may be attacked and disabled as part of an attack on the host on which the IDS is located.

The source of data used for analysis also affects the ability of the IDS to verify success or failure of an attack. In particular, a host-based IDS can determine whether an attack was successful, with greater accuracy and fewer false positives than a network-based system. This arises from the fact that a host-based IDS uses logs that contain events that have actually occurred. A network-based IDS can only determine that an attack was initiated, but it cannot determine whether it was successful. This implies that, after a network-based IDS has detected an attack, administrators must manually investigate each attacked host to determine whether it was attacked.

The detection capability of an IDS is also affected by the environment in which an IDS is used. In particular, the use of encryption of network traffic and the use of switches in a network may limit the detection capability of a network-based IDS.

A network-based IDS cannot analyse encrypted packets, depending on where the encryption resides within the protocol stack. Therefore, a network-based IDS will not detect certain attacks, which could be detected by a host-based IDS if the incoming data stream has been decrypted at the operating system level.

Switches allow large networks to be managed as many smaller network segments, and provide dedicated links between hosts serviced by the same switch. This may present difficulties in choosing the best locations for deploying a network-based IDS to achieve sufficient network coverage. Although traffic-mirroring and administrative ports on switches can help, these techniques are not always appropriate. A host-based IDS has greater visibility in a switched environment because it can reside on as many critical hosts as is required.

Furthermore, a host-based IDS resides on the host that it monitors. Therefore, although no additional hardware is required, a host-based IDS uses the resources of the monitored host, affecting the host's performance.

A network-based IDS, on the other hand, usually only examines network traffic without interfering with the normal operation of the network. However, a network-based IDS may have difficulty in processing all packets in a network with high volumes of traffic. The inability to process packets quickly enough may result in some packets being dropped. Therefore, the IDS may fail to recognize an attack launched during periods of high traffic volume. Some vendors are attempting to solve this problem by implementing the IDS completely in hardware, which will result in faster packet processing. Other vendors attempt to solve this problem by trying to detect fewer attacks and to also detect attacks with as little computing resources as possible, which may reduce the overall effectiveness of the IDS.

Once the required data has been obtained, it is analysed for possible intrusions. The way in which the collected data is analysed is determined by the intrusion detection model used by the IDS.

2.3.2 The intrusion detection model used by the IDS

With respect to the model of intrusion detection, intrusion detection systems are classified according to what analysis technique is used to detect intrusions. IDSs mainly use two techniques: anomaly detection and misuse detection (Bace, 2000; Bace & Mell, 2001). It is also possible to use a combination of both anomaly and misuse detection techniques (Botha, 2004; Anderson, Frivold & Valdes, 1995).

2.3.2.1 IDSs based on anomaly detection

An IDS that employs anomaly detection first creates a profile of normal system or user behaviour, during which no intrusion takes place. Once the profile has been created, the IDS analyses the current behaviour of the system with the behaviour recorded earlier. It is assumed that any intrusive actions will result in behaviour that is different from behaviour that is normally seen in the system. Any significant deviations from the normal behaviour are treated as intrusive behaviour.

The allowable deviation from the normal behaviour is defined as a threshold set by the specific detection method used or by the system security administrator. Selecting an appropriate threshold level is critical for the proper detection of intrusions. If the allowable deviation from the normal behaviour is small, then many anomalous activities that are not intrusive will be misdiagnosed as being intrusive. That is, the false positive (false alarm) rate will be high. On the other hand, if the allowable deviation from the normal behaviour is large, then some intrusive activities may not be recognized as being sufficiently anomalous for the IDS to treat them as intrusions. This results in a false negative, because the IDS falsely reports that no attack has occurred (Sundaram, 1996).

An IDS based on anomaly detection can detect attacks that the IDS has not seen before. Therefore, unlike IDSs based on misuse detection, it does not need to be updated with new attack signatures when new attacks are devised.

IDSs based on anomaly detection usually have a high false positive rate. This results from the fact that users and systems often exhibit legitimate but previously unseen behaviour. The previously unseen behaviour is different from that which is considered as normal by the IDS and is therefore treated as anomalous and as a possible intrusion. In addition, anomaly detection approaches often require extensive "training sets" of system event records to allow them to characterize normal behaviour patterns.

2.3.2.2 IDSs based on misuse detection

An IDS that employs misuse detection is based on searching for attack signatures in the behaviour of the system and its users. An attack signature is a known attack method that is known to exploit system vulnerabilities and cause security problems.

Misuse detection is very effective at detecting known attacks, which leads to a low number of false alarms. Moreover, because known attacks can be detected, misuse detection can reliably diagnose the use of a specific attack tool or technique. Therefore, system administrators, regardless of their level of security expertise, can track security problems on their systems and can initiate appropriate incident handling procedures to deal with the specific attack (Bace & Mell, 2001).

An IDS that employs misuse detection cannot detect previously unseen attacks because it only knows how to detect those attacks for which signatures have been defined. Therefore, when new attacks are devised, the IDS must be updated with new attack signatures. In addition, if an attack signature is too specific for a particular attack method, then the IDS will not be able to use that signature to detect variants of that attack method.

2.3.2.3 Anomaly detection approaches

Biermann, Cloete and Venter (2001) identify the following approaches to anomaly detection:

Statistical approach

In this approach, the normal, or expected, behaviour is defined by collecting data relating to the behaviour of legitimate users over a period of time. Statistical tests are then applied to the observed behaviour to determine the legitimacy of the behaviour.

The current behaviour and the original learnt behaviour may be merged at intervals, which allows the IDS to adaptively learn the behaviour of users. However, it also allows the IDS to be gradually "trained" by intruders to recognize intrusive events as normal behaviour (Sundaram, 1996).

Predictive pattern generation

In this approach, future events are predicted based on events that have already occurred. The IDS generates rules that define the probability that a certain event will occur. A rule consists of a left-hand side and a right-hand side. The left-hand side defines two concurrent events, while the right-hand side provides the probability of a specific event following the events defined on the left-hand side. If the left-hand side of a rule is matched, but the right-hand side statistically deviates from the prediction, then the event is treated as intrusive.

This approach has several benefits (Sundaram, 1996). Firstly, anomalous activities that were difficult to detect with traditional methods can be detected using rule-based sequential patterns. Secondly, systems based on this model are highly adaptive to changes, while also making it easier to detect users who try to "train" the system during its learning period to recognize anomalous behaviour. Lastly, this approach allows anomalous activities to be detected and reported within seconds of receiving audit events.

Neural networks

Neural networks use adaptive learning techniques for characterization of anomalous behaviour. This approach operates on historical sets of training data. These sets of training data should not have any data that indicate intrusions or other undesirable user behaviour (Bace, 2000).

A neural network consists of numerous simple processing elements called *units* that interact by using weighted *connections*. The knowledge of a neural network is represented in the structure of the network, in terms of the connections between different units and their weights. The actual learning process takes place by adding and removing connections and by changing weights.

Neural network processing has two stages. During the first stage, the network is populated by using a training set of historical or other sample data that represents user behaviour. During the second stage, the neural network accepts event data and compares it to historical behaviour references to determine similarities and differences.

The neural network indicates that an event is abnormal by changing the state of the units, by changing the weights of connections and by adding or removing connections. Stepwise corrections are also performed by the network to modify the definition of what constitutes a normal event.

Sequence matching and learning

This approach, introduced by Lane and Brodley (1997), applies machine learning to anomaly detection. An IDS that uses this approach is based on the assumption that a user responds in a predictable manner to similar situations, which leads to repeated sequences of actions. The characteristic sequences of actions generated by users can be learnt to create a user profile. The differences in characteristic sequences are used to distinguish a valid user from an intruder masquerading as that specific user.

2.3.2.4 Misuse detection approaches

Approaches to misuse detection include the following (Biermann, Cloete & Venter, 2001):

Expert systems

In this approach, past intrusions, known system vulnerabilities and the security policy are encoded as rules. As information is gathered during operation of the IDS, the expert system determines whether any rules have been satisfied (Frank, 1994).

Keystroke monitoring

This approach monitors keystrokes for attack patterns. As this method does not analyse the running of a program, but only the keystrokes, a malicious program cannot be flagged for intrusive activities (Sundaram, 1996).

Model-based

Model-based approaches are based on modelling known intrusion attempts as sequences of user behaviour. The behaviours are then modelled as events in an audit trail. To detect intrusions, the IDS determines how the identified user behaviour is manifested in an audit trail.

State transition analysis

In this approach, a state transition diagram is used to graphically represent the actions performed by an intruder to compromise a system. An intrusion is viewed as a sequence of actions performed by an intruder that lead from some initial state on a computer system to a target compromised state. State transition analysis diagrams identify the requirements and the compromise of the penetration. They also show the essential actions that need to occur for an intrusion to be successfully completed.

Pattern matching

The pattern matching approach is based on encoding known intrusion signatures as patterns that are matched against the audit data. An IDS based on pattern matching attempts to match incoming events to the patterns that represent intrusion scenarios. An event consists of monitored changes in the state of the system, or part of the system. It can also represent a single action by a specific user, an action by the system, or a series of actions resulting in a single, observable record.

The intrusion detection model, or the data analysis technique, used by an IDS to detect intrusions, is implemented at one or more locations in the computer system. The number of the locations at which data is analysed is determined by the way in which the IDS components are distributed.

2.3.3 The distribution of the IDS components

With respect to component distribution, intrusion detection systems are classified according to the way in which their components are distributed. Spafford and Zamboni (2000) identify two such classes: centralized and distributed intrusion detection systems. A centralized IDS analyses its data at a fixed number of locations. These locations are independent of the number of hosts being monitored. A distributed IDS analyses its data at a number of locations proportional to the number of hosts that are being monitored. In this classification, only the locations and the number of the data analysis components are considered; the data collection components are not considered.

A centralized IDS does not scale well. It only has a fixed number of analysis components. Therefore, if the number of monitored hosts increases, then the analysis components will need more computing and storage resources to support the load. A distributed IDS, on the other hand, can add components as needed, making it easier to scale to a large number of hosts. However, scalability of a distributed IDS can be limited by the need to communicate between the components and by the need for central co-ordinating components.

A centralized IDS typically has fewer analysis components than a distributed IDS. This makes a centralized IDS easier to configure globally, but it may be difficult to adjust for specific characteristics of the different hosts being monitored. The smaller number of analysis components in a centralized IDS also implies that the analysis information is located in fewer locations, which allows for easier detection of changes in global behaviour. A distributed IDS has its analysis information distributed, which may make it more difficult to adjust to global changes in behaviour, although local changes are easier to detect.

2.4 Components of an IDS

The Common Intrusion Detection Framework (CIDF) defines several components that are found in any IDS. The different components are discussed below (CIDF, 1999; Lundin & Jonsson, 2002).

2.4.1 Event generators

Also known as "E-boxes", event generators serve as the data collection components of an IDS. They collect data about the events they are specialized to notice. The type of event data collected depends on what type of attacks the IDS must be able to recognize. They can also perform some pre-processing of data, for example to transform data to a common format, or to make a preliminary filtering of the data.

2.4.2 Event databases

Event databases, or "D-boxes", store information produced by the event generators and analysis engines for later use. An example of an event database is a database that stores system log data.

2.4.3 Analysis engines

Analysis engines, or "A-boxes", implement the detection algorithm. They analyse the data that is stored in the event databases, which has been collected by the event generators.

Analysis engines also make use of two other databases: the detection policy database and the state information database (Lundin & Jonsson, 2002). The detection policy database contains pre-programmed information about how to detect intrusions, such as the different intrusion signatures and thresholds. The state information database is used to store dynamic information for detection, such as state information about partially fulfilled intrusion signatures and other information about current behaviour in the system.

2.4.4 Response units

Information about events that have been classified as anomalous or intrusive by the analysis engines is sent to the response units. Response units, or "R-boxes", decide how to respond to different events, based on pre-programmed rules in the response policy database. Different parameters, such as the certainty connected to the event and the potential impact of the event, affect what response action will be taken.

The type of response taken can be classified as active or passive (Bace, 2000). Active responses involve taking an action after the detection of an intrusion. Examples of such actions include changing the data collection configuration or the detection policy to collect more information about an event in progress, changing the environment, or taking some action against the intruder. Passive responses, on the other hand, do not

take any action other than to report and record the intrusion. This approach assumes that the user of the IDS will be able to take the necessary action to respond to a detected intrusion.

An outline of a generic IDS is shown in Figure 2.1 (based on Lundin and Jonsson, 2002).

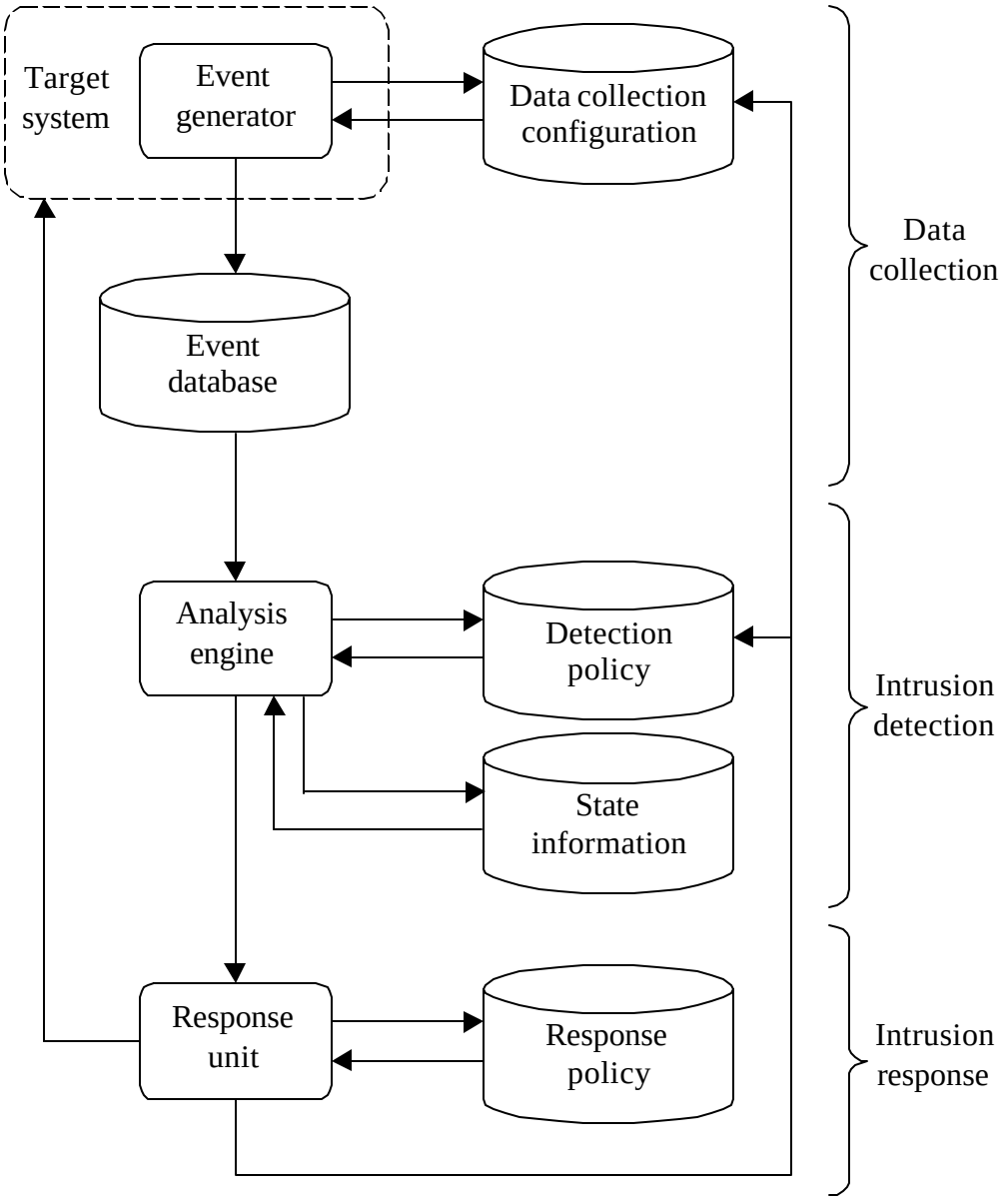


Figure 2.1 Components of a generic IDS (based on Lundin and Jonsson (2002): figure 1)

These components may be organized in different ways, resulting in different IDS architectures. These architectures are discussed in the next section.

2.5 IDS architectures

The discussion in this section is based on the work of Jansen (2002), Jansen et al. (2000) and Jansen et al. (1999). Therefore, references to these sources will be omitted in this section.

2.5.1 Early IDS architectures

The first generation of intrusion detection systems followed an architecture that only had two logical components: the data collection component and the analysis component. The data collection component obtains information from either audit logs and internal interfaces at the host (Lunt & Jagannathan, 1988; Sebring et al., 1988; Smaha, 1988) or from monitoring packets on attached networks (Heberlein et al., 1990). The information is then passed to a centralized analysis component that analyses it for intrusions. Such an architecture is shown in Figure 2.2. The two logical components may be located either on a single host or on separate hosts.

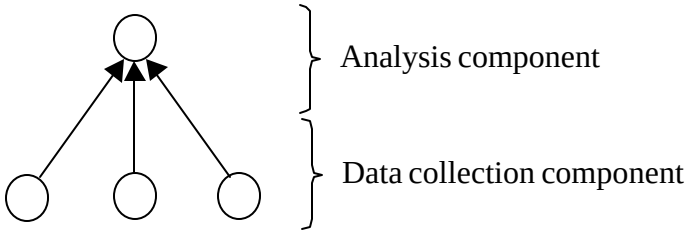


Figure 2.2 Two-component architecture of an IDS

This type of architecture is effective for small collections of monitored hosts. However, the centralized analysis limits the system's scalability: as more collection components are added, the single analysis component must analyse more information. Later generations of IDSs provide scalability by introducing intermediate components between the collection and analysis components to form a hierarchy. These components pre-process and combine collected information into input for analysis.

2.5.2 Hierarchical IDS architectures

Nearly all present-day commercial intrusion detection systems follow a hierarchical architecture (Jansen, 2002). Hierarchical architectures are used because they are excellent for creating scalable distributed IDSs with central points of administration. Such an architecture follows a tree structure as shown in Figure 2.3. Individual nodes within a network are shown with circles and the information flows between different types of nodes are shown with arrows.

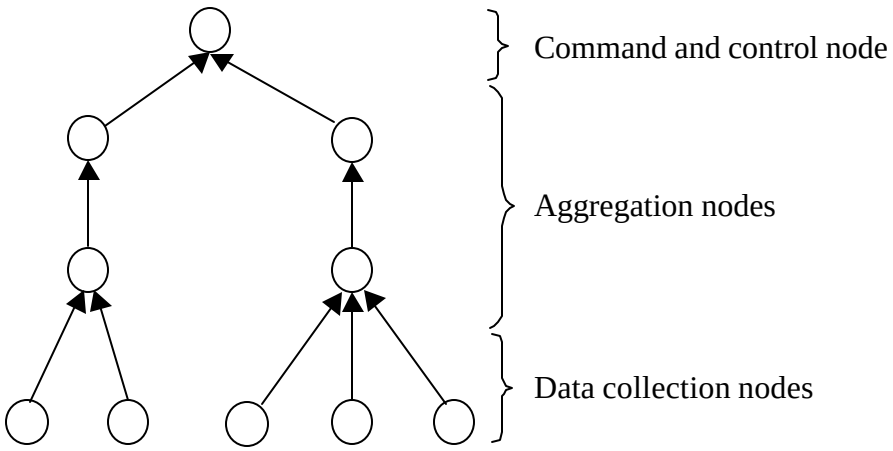


Figure 2.3 Hierarchical architecture of an IDS

The leaf nodes represent network-based or host-based collection points at which information is gathered. The event information is passed to internal nodes, which aggregate information from multiple leaf nodes. Further aggregation, abstraction and data reduction occur at higher internal nodes until the root node is reached. The root node is a command and control system that is responsible for detecting intrusions and for issuing responses. Typically, the root node also reports to an operator console where an administrator can manually assess status and issue commands to the IDS.

Hierarchical architectures generally provide efficient communication, whereby refined information filters upward in the hierarchy and control downward. Although such an architecture is excellent for creating scalable distributed IDSs with central points of

administration, it is rather rigid because of the tight binding between functionality and lines of communication that tend to evolve.

Reliance on hierarchical structures for components makes the IDS vulnerable to direct attack. Many single points of failure exist in an IDS that has no redundant communication lines or the capability to dynamically reconfigure relationships in the case of failure of key components. For example, an attacker may interrupt the operation of the entire IDS by successfully disabling the root node. Since the root node is responsible for detecting intrusions and for issuing responses, disabling it will not allow the IDS to detect intrusions.

An intruder who knows that an IDS is used may want to disable it. By disabling the IDS, the intruder can attack the rest of the computer system undetected. Therefore, the critical role played by the central controller makes it a likely target of attack. Although such critical components usually reside on platforms that have been hardened to resist direct attack, the IDS may still be vulnerable as other survivability techniques such as redundancy, mobility, or dynamic recovery are lacking in current implementations. A system could also employ redundant components for each key node to avoid this problem. However, such a solution does not offer much fault-tolerance because a determined and knowledgeable attacker can disable a small number of backups.

2.5.3 Network IDS architectures

In contrast to a hierarchical IDS architecture, network IDS architectures allow information to flow from any node to any other node. Network architectures tend to be inefficient in communication because of the unconstrained communication flow. This is compensated for with their flexibility to function.

This type of architecture is used by Cooperating Security Managers (White, Fisch & Pooch, 1996). The data collection, aggregation, as well as the command and control functions are consolidated into a single component located on every monitored system. Any significant events that occur at one system that stem from a connection originating

at another are reported to the system manager of the originating system. This reporting is performed by the security manger at the system where the event occurred.

Although the components of such an IDS implicitly tend toward a hierarchy, the tendency is not strict, since communication can occur between any type of components and not strictly on a one-to-one or master / slave basis. For example, a data collection unit could directly notify the command and control unit of a critical event, rather than through an aggregation node.

2.5.4 Hybrid IDS architectures

A hybrid IDS architecture is a combination of both hierarchical and network IDS architectures. It has an overall hierarchical architecture, although it has no distinct root node. It also follows a network architecture by allowing its components to communicate outside the strict hierarchy when useful. An example of such an architecture is shown in Figure 2.4.

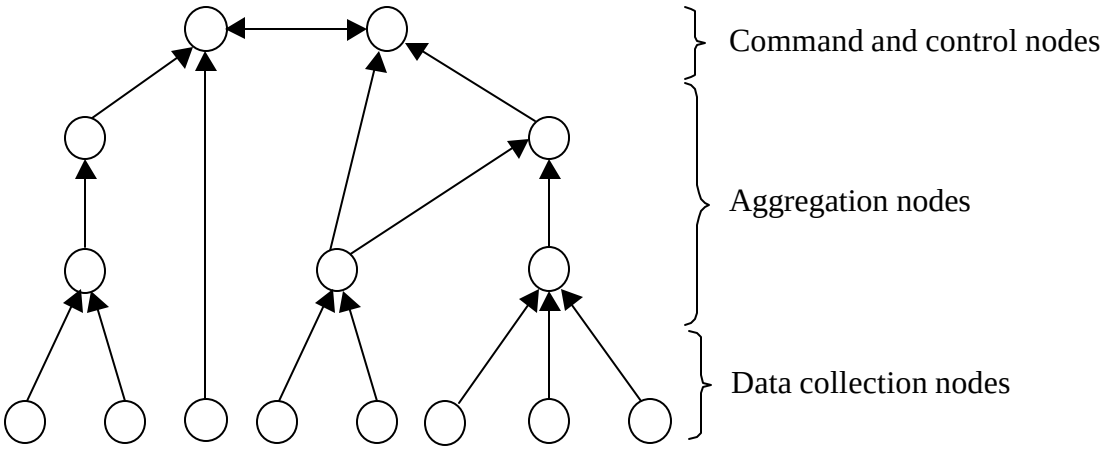


Figure 2.4 Hybrid architecture of an IDS

The architecture in Figure 2.4 shows a peer relationship between the command and control nodes. It also shows direct communication between a data collection node and a command and control node. This type of communication may be useful to communicate

an important or critical event. There is also a redundant communication link, for fault-tolerance, between a command and control node and an aggregation node.

2.6 Desirable characteristics of an IDS

There are several characteristics that are desirable in an IDS. Jansen et al. (1999) have divided these characteristics into two groups of requirements: functional requirements and performance requirements.

2.6.1 Functional requirements

The common functional requirements of intrusion detection system include the following:

- The IDS must continually monitor and report intrusions.

- When an intrusion occurs, the IDS must supply enough information to determine the extent of the damage, establish responsibility for the intrusion, and allow the system to be repaired.

- As each host and network segment will require their own tests, and these tests will need to be continuously upgraded (and eventually replaced with new tests), the IDS should be modular and configurable.

- The IDS should be easily and frequently updated with new attack signatures as new security advisories and security patches become available and as new vulnerabilities and attacks are discovered.

- The IDS itself is a primary target of attack because of its assigned critical role of monitoring the security state of the network. Therefore, the IDS should exhibit a high degree of fault-tolerance and should provide graceful degradation to enable it to operate in a hostile environment.

- The IDS should adapt to changes in network topology and configuration as computer devices are dynamically added and removed from the network.

- Anomaly detection systems should have a very low false alarm rate. It may not be sufficient to decrease the percentage of overall false alarms since their absolute number may increase with increased network connectivity and traffic.

- The IDS should be able to learn from past experiences and improve its detection capabilities over time.

- To respond to various attacks, system administrators will need to use decision support tools. Therefore, the IDS will be required not only to detect anomalous events, but also to take automated corrective action.

- It should be possible for the IDS to perform data fusion and process information from multiple and distributed data sources.

- Data reduction tools will need to help the IDS process the information gathered from data fusion techniques. Data mining tools will be helpful in running statistical analysis tools on archived data in support of anomaly detection techniques.

- Due to rapid changes in network conditions and limited network administration expertise, it is difficult for system administrators to diagnose problems and take corrective action to minimize the damage that intruders can cause. Therefore, the IDS should have the capability to provide an automated response to suspicious activities.

- It will become necessary to detect and react to distributed and co-ordinated attacks. Co-ordinated attacks against a network will be able to assemble greater forces and launch many more and varied attacks against a single target. By rapidly evolving, these attacks can be permutations of known attacks and be launched at little cost to the attackers.

- As no vendor toolset is likely to excel in or provide complete coverage of the detection, diagnosis, and response capabilities, the IDS should be able to work with other commercial off-the-shelf security tools. The IDS framework should be able to integrate various data reduction, forensic, host-based and network-based security tools. The value of the IDS will be further increased by interoperability and conformance to standards.

- After an intrusion has been detected, the IDS data often requires additional analysis to assess any damage to the network. Although the anomalous event was the first detected, it may not be the first attempt to gain unauthorized access to the network. Therefore, before the network can be restored to a safe condition, post-event analysis will be needed to identify compromised machines.

- The IDS itself must also be designed with security in mind. It should not introduce additional vulnerabilities. It should also be able to authenticate the administrator and audit administrator actions. The IDS data should also be protected and the IDS devices should be authenticated.

2.6.2 Performance requirements

Although an IDS may be functionally correct, it is of little use if it detects attacks too slowly. The IDS performance requirements include the following:

- To the extent possible, any anomalous events or breaches in security should be detected in real-time and reported immediately. This may minimize the damage to the network and the loss or corruption of data.

- The IDS should not impose a large overhead on the computer system. There is a trade-off between additional levels of security monitoring and performance.

- The IDS should be scalable to enable it to handle additional computational and communication load, as new computer devices are added to the network.

2.7 Current shortcomings of intrusion detection systems

Current IDSs have many shortcomings. Although developers continue to address the deficiencies by improving and refining existing techniques, some deficiencies are inherent in the way IDSs are constructed. The most common shortcomings include the following (Jansen, 2002):

- *Lack of efficiency*. Current IDSs are not efficient enough to evaluate events in real-time in computer systems with a large number of events and on high-speed networks with large volumes of traffic. Consequently, host-based IDSs often slow down a system, while network-based IDSs drop network packets that they do not have time to process. In addition, as new attacks are devised, the IDS must be updated to discover them. Considering that old attacks are not removed from the IDS, more processing time is required by the detection algorithm for greater attack coverage.

- *High number of false positives*. Current IDSs have a high false positive rate because recognition of intrusions is not perfect. Changing thresholds to reduce false alarms raises the number false negatives (true attacks that are not detected).

- *Burdensome maintenance*. To correctly configure and maintain an IDS often requires special knowledge and substantial effort.

- *Limited flexibility.* IDSs have typically been written for a specific environment and have proved difficult to use in other environments that may have similar policies and concerns. It can also be difficult to adapt the detection mechanism to different patterns of usage. Tailoring detection mechanisms specifically to the system in question and replacing them over time with improved detection techniques is also problematic with many IDS implementations. It is often necessary to restart the IDS to allow changes and additions to take effect.

- *Vulnerability to direct attack*. Many IDSs are vulnerable to direct attack because they rely on hierarchical structures for components. A control branch of the IDS can be cut off by an attacker who attacks an internal node. Moreover, the entire IDS can

be disabled if the root command and control node is disabled. Although such critical components usually reside on platforms that have been hardened to resist direct attack, current implementations lack other survivability techniques such as redundancy, mobility, or dynamic recovery.

- *Vulnerability to deception*. A network-based IDS evaluates network packets by using a generic network protocol stack to model the behaviour of the protocol stack of the hosts that it is protecting. An attacker may take advantage of this discrepancy by sending, to a target host, packets that have been specially adapted to be interpreted differently by the IDS and by the target host. This can be done in various ways, such as altering fragmentation, sequence number, and packet flags (Ptacek & Newsham, 1998). The attacker is able to attack the target while the IDS remains blind to the attack or is fooled into interpreting that the target resisted the attack.

- *Limited response capability.* Although intrusion detection is important, many times a system administrator is not able to immediately analyse the reports from an IDS to take appropriate action. This gives an intruder time before being countered by the actions of the administrator. Many IDSs have begun to provide automated response capabilities to reduce the time available for an intruder. However, their ability to adapt dynamically to an attack is limited.

- *No generic building methodology*. There is no structured methodology available to build an IDS. Consequently, the cost of building an IDS from available components is high.

2.8 Conclusion

This chapter provided theoretical background information on intrusion detection systems. Intrusion detection systems were discussed in terms of their need, classification, components, architectures, desirable characteristics for an IDS, as well as the current shortcomings of IDSs. The next chapter provides theoretical background information on mobile agents.

CHAPTER 3

MOBILE AGENTS

3.1 Introduction

A *software agent* can be defined as (Bradshaw, 1997) "... a software entity which functions continuously and autonomously in a particular environment ... able to carry out activities in a flexible and intelligent manner that is responsive to changes in the environment ... Ideally, an agent that functions continuously ... would be able to learn from its experience. In addition, we expect an agent that inhabits an environment with other agents and processes to be able to communicate and co-operate with them, and perhaps move from place to place in doing so."

Software agents can be static or mobile. Stationary agents remain resident on a single system during their execution lifetime, executing only on the system where they have began execution. Mobile agents, on the other hand, have the ability to migrate from one system in a network to another during their execution lifetime.

This chapter discusses mobile agents and is organized as follows. In Section 3.2, various network computing paradigms are discussed, showing how the mobile agent paradigm differs from other network computing paradigms. Section 3.3 discusses the fundamental elements of a mobile agent system. This is followed by four sections, which contain a discussion of the essential behaviour of a mobile agent: creation, disposal, migration and communication. The potential benefits of applying mobile agents in intrusion detection are discussed in Section 3.8. In Section 3.9, the potential drawbacks of applying mobile agents to intrusion detection are discussed. This is followed by a discussion of the different security issues related to the use of mobile agents. This chapter is concluded in Section 3.11.

3.2 Network computing paradigms

There are various network computing paradigms that support communication between entities in a distributed computer system. This section contains a discussion of these paradigms to show how the mobile agent paradigm differs from other network computing paradigms (based on Lange and Oshima (1998), and Rothermel and Schwehm (1998)). Figure 3.1 on page 43 shows the various degrees of mobility offered by the different paradigms that are discussed below.

3.2.1 Message passing

Message passing was the first network communication paradigm proposed. It allows processes to communicate by explicitly sending and receiving messages. Due to its flexibility, the concept of message passing can support many varieties of communication patterns. However, developing distributed applications based on message passing primitives is complex and error-prone.

3.2.2 Client-Server

The client-server paradigm was proposed to provide a higher-level communication abstraction than message passing. It allows processes to communicate by requesting services from other processes, rather than by explicitly sending and receiving messages. A client process can issue requests for certain services to a server process, which provides the service and returns the results to the client process.

This paradigm is widely known; most distributed systems have been based on it. It is supported by technologies such as remote procedure calls (RPC), object request brokers (CORBA), and Java remote method invocation (RMI).

The client-server paradigm provides a more convenient form of interprocess communication than message passing. For example, with RPC, much of the complexity associated with reliable communication is hidden in so-called stub procedures. Stub procedures can provide methods for marshalling and unmarshalling of messages,

encoding and decoding in heterogeneous environments, and failure recovery. However, processes can communicate only through calling of and receiving the result of a remote procedure; processes are restricted to communicating by requesting and receiving services. Therefore, the client-server paradigm does not provide as much flexibility as message passing.

3.2.3 Remote execution

In the client-server paradigm, when the client requests a service from a server, it is the server that executes the necessary code and provides the results to the client. In many cases, however, it is desirable to send code (e.g. a procedure) to a remote node and execute it there. The remote execution paradigm provides for such situations and it allows one node to send code to another node for execution.

Two distinct types of remote execution exist: remote evaluation and code-on-demand. In remote evaluation, a client node transfers code (and the necessary data) to a server node, which executes the code. This is a so-called "push" approach, because a service is uploaded to a computer. Code-on-demand is the opposite of remote evaluation; a server node transfers code (and the necessary data) to a client node. Code-on-demand is a so-called "pull" approach, because a service is downloaded from a computer. Remote execution provides a flexible way to dynamically extend the behaviour of servers and clients, through remote evaluation and code-on-demand, respectively.

Examples of this paradigm are Java applets and servlets. Java applets are downloaded in web browsers and executed locally. Java servlets, on the other hand, are uploaded to remote web servers and executed there.

3.2.4 Mobile agents

Remote execution transfers only the code and the necessary data to a remote location; it does not transfer an executing program. The concept of mobile agents allows an executing program to migrate from one node to another in a computer network. For this

to occur, both the code and the state information (also known as the agent state; see Section 3.3.1.1) of the agent have to be transferred during agent migration.

We should note that the concept of agent migration is not the same as process migration. The main difference between the two concepts lies in what entity decides when migration takes place and to which destination node in the network. In the case of process migration, the operating system makes this decision and migration is transparent to the process. In the case of agent migration, the agent itself decides when and where to migrate.

Existing mobile agent systems provide two forms of migration: strong migration and weak migration (Rothermel & Schwehm, 1998):

3.2.4.1 Strong migration

In strong migration, the agent system captures the entire state of the agent (i.e. the data and the execution state). The state and the code of the agent are then transferred to the destination. Once the agent is received at the destination, its state is automatically restored.

In this scheme, the capture, transfer and restoration of the complete agent state is done transparently by the underlying agent system. However, to provide this degree of transparency in a heterogeneous environment will require at least a global model of the agent state and a transfer syntax for this information. In addition, the agent system must provide functions to externalize and internalize agent state. Strong migration might also be a time-consuming operation, since the complete agent state can be large, especially for multithreaded agents.

3.2.4.2 Weak migration

In weak migration, the agent system does not capture the entire state of the agent, but only its data state. The size of the data state can also be limited by allowing the programmer to select the variables that make up the agent state. Therefore, only the data state and the code of the agent are transferred to the destination.

In this scheme, the programmer must encode the agent's relevant execution states in the program variables. The programmer must also specify a start method, which will decide where the agent should continue its execution after migration (based on the encoded state information). Therefore, although weak migration reduces the amount of information that must be communicated, it puts an additional burden on the programmer and makes agent programs more complex.

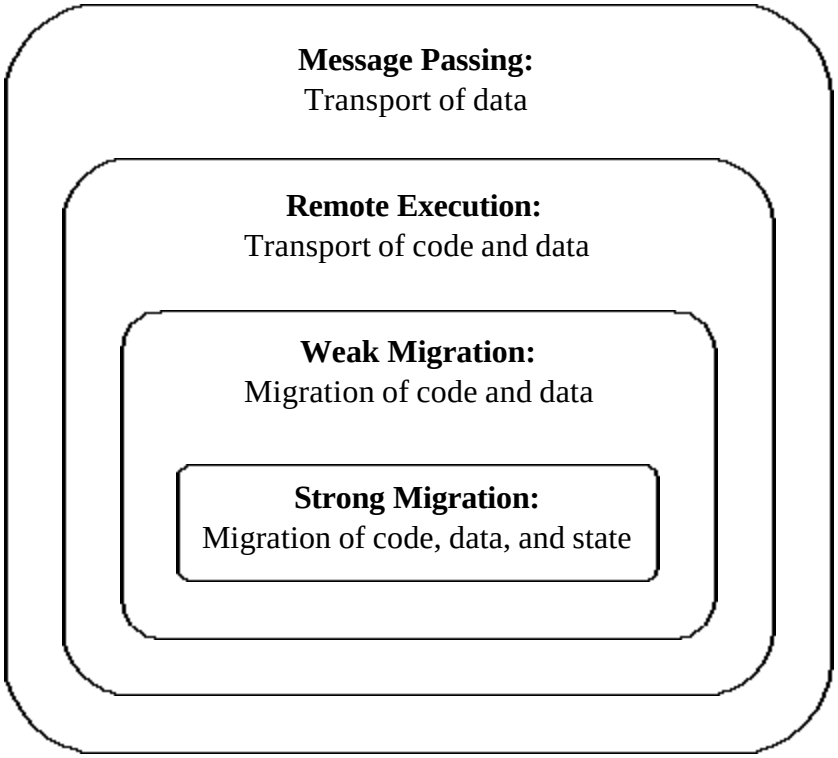


Figure 3.1 Degrees of mobility (based on Rothermel and Schwehm (1998): figure 2)

3.3 The elements of a mobile agent system

There are two fundamental concepts in the mobile agent model: the agent and the place. These two concepts are very briefly discussed in this section, by using the work of Lange and Oshima (1998). The reader is referred to Lange and Oshima (1998) for more detail on these concepts.

3.3.1 The agent

A mobile agent has five attributes: state, implementation, interface, identifier, and principals.

3.3.1.1 Agent state

The state of an agent at any time is a snapshot of its execution. It is needed by the agent to allow it to resume execution after migration. The agent's state can be divided into its execution state and its data state. The execution state is an agent's run-time state, including its program counter and frame stack. The data state is the set of values of the agent's variables.

An agent does not always need to capture and transport its entire execution state to allow it to resume its execution at the destination. In many cases, the values of the agent's variables are sufficient to allow it to determine what to do when it resumes its execution at its destination.

3.3.1.2 Implementation

The implementation of an agent is the code it uses to execute. When an agent migrates, it can take its entire implementation code to its destination, or it can go to its destination, determine what code is already there, and retrieve any missing code over the network.

An agent's implementation should be both executable at the destination host and be safe for the host to execute (i.e. in a way that ensures the confidentiality, integrity, and availability of the host, its data and services). Platform independence can be provided through scripting and interpreted languages. They also provide a controlled execution environment that has security mechanisms that restrict access to the host's private resources.

3.3.1.3 Interface

The interface of an agent allows for other agents and systems to interact with it, using an agent communication language (ACL). An agent communication language allows agents (from different vendors) to interact with one another in a multi-agent system.

3.3.1.4 Identifier

Every agent has an identifier that is unique during its lifetime and which is assigned by the place at which the agent is created. An identifier is used for recognition and location of travelling agents.

3.3.1.5 Principals

The principals of an agent are needed to determine the legal and moral responsibility of the agent. A principal is an entity whose identity can be authenticated by any system that the principal may try to access. A principal can be an individual, an organization, or a corporation.

For agents, there are at least two main principals: the manufacturer and the owner. The manufacturer refers to the author, or the provider of the agent implementation. The owner is the principal that has the legal and moral responsibility for the agents' behaviour.

3.3.2 The place

A place provides a uniform set of services that agents can make use of. It provides a context in which an agent can execute and it can contain multiple agents. A place is also sometimes referred to as an agent platform.

A place itself cannot execute agents. To do that, an agent must reside inside an engine, which serves as a virtual machine for places and their agents. An engine also provides agents and places with links to the resources provided by the host and the underlying

network. Any computer in the network can have multiple engines and each engine can have multiple places.

The location of a place is determined by a combination of the name of the place and the network address of the engine in which that place resides. This location will typically have an IP address and a port number of the engine with a place name attribute.

A place also requires principals. It identifies the person or organization for which the place acts (place master) as well as the manufacturer of the place. The manufacturer is the author, or provider, of the place implementation. The place master refers to the principal that has the responsibility for the operation of the place.

Figure 3.2 shows how the concepts of agents, places and engines relate to one another.

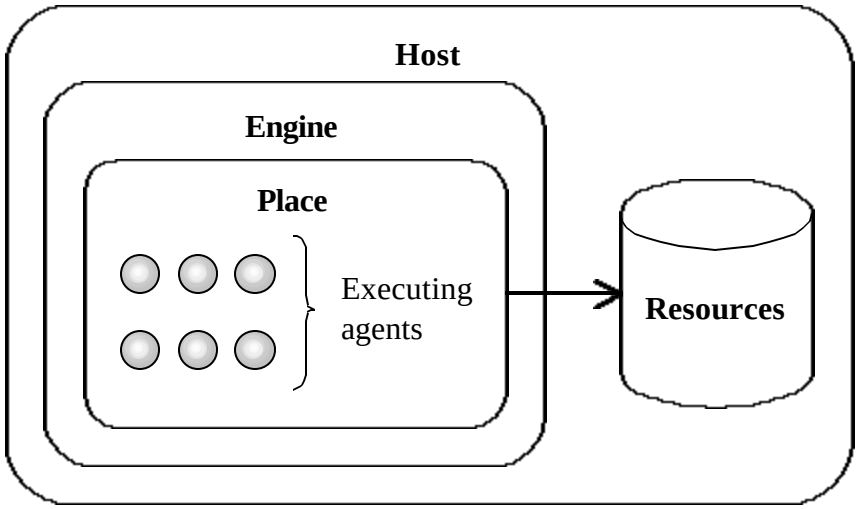


Figure 3.2 Place and engine (based on Lange and Oshima (1998):20)

In the following three sections, essential behaviour of mobile agents is discussed by using the work of Lange and Oshima (1998). Therefore, references to this source will be omitted in the next three sections.

3.4 Agent creation

An agent is created in a place. The creation of an agent can be initiated either by another agent residing in the same place or by another agent or non-agent system outside of the

place. Before an agent is created, its creator must authenticate itself to the place and establish the authority and credentials that the new agent will posses. The code of the agent must be present on the local host or a remote host. The code can also be supplied by the creator, together with the required initialization arguments.

Three steps are required to create an agent:

1. *Instantiation and identifier assignment.* The code of the agent (e.g. a class definition) is provided. The agent is also assigned a unique identifier by the place at which it is created.

2. *Initialization.* The agent is initialized by using the initialization arguments that have been provided by the creator. Once the initialization has been completed, the agent is fully and correctly installed in the place.

3. *Autonomous execution.* After being fully installed in the place, the agent begins executing.

3.5 Agent disposal

An agent can be disposed of for several reasons. For example, the agent's lifetime may have ended; no one refers to or uses the agent; or the system is shutting down. The disposal of an agent can be initiated by the agent itself, by other agents residing in the same place, or by another agent or non-agent system outside of the place. An agent is disposed of in a place.

Two steps are required to dispose of an agent:

1. *Preparing for disposal.* The agent is allowed to finalize its current task before it is disposed of.

2. *Suspension of execution.* The place suspends the execution of the agent.

3.6 Agent migration

Although the process of agent migration is initiated by the agent itself, the actual transfer of the agent is managed by the origin place (the current place of the agent) and by the destination place (the intended receiving place).

There are two stages in transferring an agent: dispatching the agent and receiving the agent (Figure 3.3).

3.6.1 Dispatching an agent

Before a mobile agent migrates, it must identify its destination. If the agent does not specify the place, then the agent will run in a default place, as selected by the destination agent system. Once the location of the destination has been determined, the mobile agent informs the local agent system that it wants to migrate to the destination agent system. This message is sent through an internal API (Application Programming Interface) between the agent and the agent system. When the agent system receives the agent's request to migrate, it performs the following tasks:

1. *The agent is suspended.* The agent system warns the agent that it is about to be transferred. The agent is allowed to prepare for the transfer (e.g. complete its current task) and then its execution thread is halted.

2. *The agent is serialized.* The state and the code of the agent are serialized. That is, a persistent representation of the state and the code of the agent is created so that it can be transported over a network.

3. *The agent is encoded.* The serialized agent is encoded for the chosen transport protocol.

4. *The agent is transferred.* The engine establishes a network connection to the specified destination, and the encoded serialized agent is transferred.

3.6.2 Receiving an agent

Before an engine will receive an agent, it will determine whether it can accept an agent from the sending host. The actual data transfer will take place only after the sender has successfully authenticated itself to the receiving engine. The following tasks are performed by the receiving agent system:

1. *The agent is received.* Once the destination engine agrees to the transfer, the encoded serialized agent is received.

2. *The agent is decoded.* The incoming data stream is decoded.

3. *The agent is deserialized.* The persistent representation of the state and the code of the agent is deserialized; the transferred agent state is restored.

4. *The agent resumes execution.* The recreated agent is notified of its arrival at the destination place. The agent prepares to resume its execution and is given a new thread of execution.

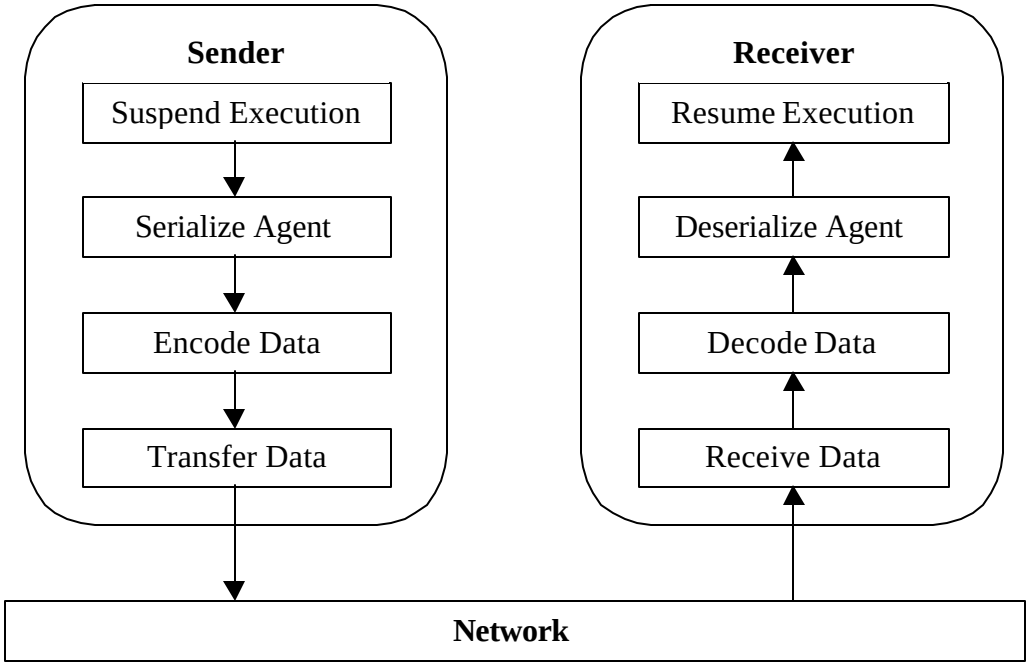


Figure 3.3 Agent migration (Lange & Oshima, 1998:23)

3.7 Agent communication

Agents have the ability to communicate with other agents that reside within the same place, or that reside in other places. In this section, the various communication types that agents may use to interact with one another are discussed.

3.7.1 Types of communication

There are several possible types of interactions that agents can take part in. These include: agent / service agent interaction, mobile agent / mobile agent interaction, agent group communication, as well as user / agent interaction (Rothermel & Schwehm, 1998).

3.7.1.1 Agent / service agent interaction

This type of interaction occurs when an agent interacts with service agents. This style of interaction is usually client-server, since the service agents export operations (or methods) that can be requested by other agents. An RPC-like communication should be incorporated in the agent system to simplify the development of agent programs.

3.7.1.2 Mobile agent / mobile agent interaction

This type of interaction occurs when a mobile agent interacts with another mobile agent in a peer-to-peer fashion. The communication patterns that may occur in this type of interaction might not be limited to request / response only. Message passing schemes are used to provide the required degree of flexibility.

3.7.1.3 Agent group communication

This type of interaction occurs when the sender does not know the identities of the agents that are interested in the sent message. Such a situation may occur when, for example, a given task is performed by a group of agents, with each agent taking over a subtask. In this case, the agent that requested the task may not know the identities of the

individual agents of the group. Therefore, communication is anonymous in the sense that the sender only knows the group rather than the individual group members. This type of communication is supported by group communication protocols, shared memory and event messages (where an agent will send out event messages anonymously, and the receivers explicitly register for those events they are interested in).

3.7.1.4 User / agent interaction

This type of interaction occurs when human users communicate with software agents.

3.7.2 Messaging schemes

Inter-agent communication can follow three different messaging schemes: now-type messaging, future-type messaging, and one-way-type messaging (Lange & Oshima, 1998).

3.7.2.1 Now-type messaging

In this scheme, messages are synchronous. This implies that, when an agent sends a message, its execution is blocked until the receiver of the message has completed the handling of the message and has replied to it. This scheme is the most commonly used messaging scheme.

3.7.2.2 Future-type messaging

In this scheme, messages are asynchronous. Therefore, when an agent sends a message, its execution is not blocked; the agent can continue executing while it waits for the reply. The sending agent retains a handle, which is used to obtain the result of the message. This messaging scheme is useful when multiple agents communicate with one another, because the sender does not need to wait until the receiver responds and sends the reply.

3.7.2.3 One-way-type messaging

In this scheme, messages are asynchronous and the sending agent does not retain a handle for such messages. Therefore, the sending agent does not expect a reply from the receiving agent. This messaging scheme is convenient when agents are allowed to interact with one another where the message-sending agents do not expect any replies from the message-receiving agents.

3.7.3 Sessions

A variety of agent systems require that the interacting agents establish a session (or a communication relationship in general) before they communicate (Rothermel & Schwehm, 1998). Sessions are required in situations where stateful interactions have to be supported, where an explicit communication relationship is a prerequisite. Sessions allow for synchronization of agents that want to meet for co-operation; they allow agents to identify other agents that are interested to meet at certain places. Furthermore, sessions allow agents to synchronize themselves on the event when all session peers are available and willing to co-operate.

An agent must explicitly agree to participate in sessions and may unilaterally terminate the session it is involved in at any time. Once a session is established, the agents can interact by remote procedure calls or by message passing. The session is explicitly terminated when all information has been communicated. While an agent is involved in a session, it should not move to another place. However, if an agent does move to another place while involved in a session, the session is implicitly terminated. This requirement simplifies the underlying communication mechanism; for example, it avoids the need for message forwarding.

A session may be an inter- or an intra-place communication relationship. In an inter-place session, agents are able to communicate from different places. An intra-place session, on the other hand, requires that all communication be local. Therefore, an intra-place session places restrictions on agents because they cannot communicate from different places, but need to be located at one place.

3.8 Advantages of applying mobile agents in intrusion detection

There are several benefits derived from applying mobile agents in distributed systems (Lange & Oshima, 1998). In this section, the benefits of mobile agents are discussed with reference to their applicability to the design of intrusion detection systems (Jansen, 2002; Jansen et al., 2000; and Jansen et al., 1999).

3.8.1 Reduction of network load

A distributed system often relies on communication protocols that involve multiple interactions to accomplish a task. This generates a considerable amount of network traffic. By using mobile agents, the interaction can be packaged and dispatched to a destination host, where the interactions can take place locally. Mobile agents can also reduce the need to transport large amounts of raw data in a network. For example, a large amount of network traffic will be generated when there are several hosts in the network that must transport large volumes of raw data for processing at some other host. Mobile agents can reduce the amount of network traffic by processing the data locally at each remote host and sending only the results. Therefore, by using mobile agents, the computation is moved to the data, instead of moving the data to the computation.

Current IDSs often need to process large amounts of data generated by network traffic monitoring tools and host-based audit logs. This data is typically processed locally. Abstracted data is often sent to other network locations where it is further abstracted and then eventually sent to a central processing node, which evaluates the abstracted results from all locations in the network. Although the data is usually abstracted before being sent out on the network, the amount of data may still place a significant communication load on the network.

The amount of data sent over the network can be reduced by dispatching mobile agents to the hosts on which the intrusion data resides. The mobile agent processes the data locally and sends only the results to the central processing node. Clearly, transferring an

agent that is smaller in size than the data that needs to be transferred reduces the network load.

3.8.2 Overcoming network latency

Network latency can be reduced by sending an agent with a sequence of service requests across the network rather than by issuing each service request by a separate remote procedure call.

An IDS, in addition to detecting intrusions, should also provide an appropriate response to intrusions to protect the computer system. A central controller can be used to send messages to the nodes within the network and issue instructions on how to respond to a detected intrusion. However, the central controller can become a bottleneck or a single point of failure if it must respond to a large number of events throughout the network and also handle its normal processing load. If the connections to this central server are slow or unreliable, the network communications are susceptible to unacceptable delays.

Mobile agents can be dispatched from a central controller to carry out operations directly at a remote place of interest. This allows them to respond, in real-time, to changes in their environment. Mobile agents can also take advantage of alternate routes around any problem communication lines. In addition to detecting and diagnosing potential network intrusions, mobile agents can also provide an appropriate response mechanism.

3.8.3 Asynchronous and autonomous execution

An IDS architecture that is co-ordinated by a central host requires reliable communication paths to the network sensors and intermediate processing nodes. The critical role played by this central controller makes it a likely target of attack.

An IDS based on mobile agents can continue to operate in the event of failure of a central controller or a communication link. Unlike message passing routines or remote procedure calls, once a mobile agent has been launched from its home platform, it can

continue to operate autonomously even if its home platform is no longer available or connected to the network. Therefore, a mobile agent's inability to communicate with a central controller would not prevent it from carrying out its assigned tasks. In addition, the agents that survive an attack may be able to reconstruct damaged components by cloning to restore lost functionality.

3.8.4 Dynamic adaptation

Mobile agents have the ability to sense their execution environment and autonomously react to changes. Multiple mobile agents are able to distribute themselves among the hosts of the network in order to maintain the optimal configuration for solving a particular problem. For example, an agent and its data can move to another computer if the computation load of its current host platform is too high and the host's performance does not meet the service expectations of the agent.

3.8.5 Operating in heterogeneous environments

Large computer networks usually have many different computing platforms and computing devices. Since mobile agents are generally computer and transport-layer independent, and dependent only on their execution environment, they provide a good approach for integrating heterogeneous systems. The ability of mobile agents to operate in heterogeneous computer environments is made possible by a virtual machine or interpreter on the host platform. However, virtual machines and interpreters offer only limited support for the preservation and resumption of an agent's execution state in a heterogeneous environment, because of differing representations in the underlying computer hardware.

Although mobile agents allow an IDS to operate in a heterogeneous environment, the tests performed by, or the tasks assigned to, mobile agents are often platform-dependent. Therefore, unless there will be a common programming interface available for intrusion-detection functions, the agents must either be restricted to a single class of host or be designed to provide heterogeneity in some other way (e.g. by dynamically loading the host-dependent code).

3.8.6 Robust and fault-tolerant behaviour

Mobile agents have the ability to react dynamically to unfavourable situations and events, which makes it easier to build robust and fault-tolerant distributed systems. For example, when a host is being shut down, the agents executing on that host may be warned (whenever possible) and given time to migrate and continue their operation at another host. Mobile agents can also clone themselves for redundancy and parallelism, or request assistance from other agents. These characteristics, when combined with asynchronous and autonomous execution, facilitate the building of robust and fault-tolerant systems.

3.8.7 Scalability

As more processing nodes are added to networks monitored by a centralized IDS, the computational load of the IDS increases. Even greater demands on these centralized architectures will be placed as the bandwidth and traffic of the network increase.

To provide IDS scalability, the computational load can be distributed by using a distributed IDS architecture. As the number of computing elements in the network increases, agents can be cloned and dispatched to new computers.

3.9 Disadvantages of applying mobile agents in intrusion detection

In spite of the numerous advantages of applying mobile agents in intrusion detection, there are also several disadvantages (Jansen, 2002; Jansen et al., 2000; Jansen et al., 1999).

3.9.1 Security concerns

The main obstacles to the widespread use of mobile agents are security concerns related to the use of mobile code. The different security concerns and countermeasures are discussed in Section 3.10.

3.9.2 Performance

One of the most challenging problems facing IDSs is improving the speed with which they can detect intrusions. System events must be processed in real-time and attacks must be detected quickly. This is more difficult to achieve as network bandwidth increases.

Mobile agent software will generally hinder rather than help an IDS's ability to rapidly process events and detect attacks. The runtime environments for mobile agents slow down an IDS based on mobile agents. This is especially apparent if mobile agents are implemented in slow interpreted languages.

3.9.3 Code size

Agents that perform IDS services may need to contain a large amount of code. The code size may be increased even more if the agents are to perform operating-system-specific tasks on multiple operating systems. The long time that is needed to transfer an agent between hosts may limit the functionality of an IDS based on mobile agents. Furthermore, greater computing and network resources will be needed for such a transfer.

3.9.4 Lack of a priori knowledge

Large enterprise networks are usually comprised of several different hardware platforms that may use different operating systems, have different configurations and run different applications. It is not simple to create lightweight agents that have a priori knowledge about how a system is configured and how data is arranged.

3.9.5 Limited exposure

The area of distributed control of mobile agent systems is not as well understood and as quite mature as, for example, the client-server computing paradigm. The autonomous behaviour of agents, involving collaboration with other agents at various network locations, creates a dynamic environment that requires new design methodologies and modelling tools to properly formulate and construct agent-based systems. This task is made difficult by the lack of mature agent design methodologies and modelling tools. This problem is likely to be overcome as commercial demand for these products increases.

3.9.6 Coding and deployment difficulties

The standard development process historically produces code with many faults. Given the added complexity of mobile agents, such as moving and cloning, IDSs based on mobile agents may be even more prone to faults. There is also a lack of mobile agent design, development, and management tools needed before any large-scale deployment of agent-based applications becomes feasible. This also hampers near-term deployment of an IDS based on mobile agents.

3.10 Security issues of mobile agents

The use of mobile agents has several security concerns that hinder the widespread use of this technology. There are four broad categories of security threats related to the use of mobile agents (Jansen & Karygiannis, 1999):

1. *Agent-to-agent*, in which an agent exploits the vulnerabilities of other agents residing on the same agent platform.
2. *Agent-to-platform*, in which an agent exploits the vulnerabilities of its platform.
3. *Platform-to-agent*, in which the agent platform compromises the agent's security.
4. *Other-to-platform*, in which external entities threaten the security of the agent platform.

3.10.1 Types of threats

There are several threats related to the use of mobile agents (Jansen & Karygiannis, 1999; Jansen, 1999).

3.10.1.1 Masquerading

Masquerading occurs when one entity claims the identity of another. This attack can occur in the following threat categories:

- *Agent-to-agent*, in which an agent attempts to disguise its true identity so as to mislead other agents. Masquerading as another agent harms both the agent that is being mislead and the agent who's identity has been assumed (e.g. the reputation of such an agent may be damaged by the masquerading agent).

- *Agent-to-platform*, in which an agent attempts to claim the identity of another agent so as to mislead an agent platform. This may allow the masquerading agent to pose as an authorized agent to allow it to gain access to services and resources of an agent platform it would otherwise be unable to access. By claiming the identity of another agent, the masquerading agent may damage the trust of the legitimate agent. An agent can also pose as an unauthorized agent in an effort to not be held accountable for some actions.

- *Platform-to-agent*, in which an agent platform claims the identity of another agent platform in an effort to deceive agents as to the platform's true identity and corresponding security domain. Once the agent is captured by the platform, the platform can access and modify the agent's code, state, and data.

- *Other-to-platform*, in which an agent on a remote platform disguises itself as another agent in an attempt to gain access to services and resources to which it is not entitled. Masquerading can also take place when a remote platform disguises itself as another platform to mislead other platforms or agents about its true identity. A

masquerading agent can also act in conjunction with a malicious agent platform to help deceive another remote platform.

3.10.1.2 Denial-of-service

A denial-of-service attack occurs when one entity's services are not accessible to entities that are authorized to use them. A denial-of-service attack can occur in the following threat categories:

- *Agent-to-agent*, in which an agent attempts to interfere with the normal execution of other agents. Such an attack may be launched by, for example, the agent repeatedly sending messages to other agents, which may place undue burden on the message handling routines of the recipient. A malicious agent can also intentionally distribute false or useless information in an effort to prevent other agents from completing their tasks correctly or in a timely manner.

- *Agent-to-platform*, in which an agent consumes an excessive amount of computing resources of the platform. The performance of the agent platform may be significantly reduced or even result in the termination of the agent platform. This attack can be launched both intentionally, by providing code to exploit system vulnerabilities, but also unintentionally as a result of programming errors.

- *Platform-to-agent*, in which an agent platform interferes with the normal execution of an agent by, for example, ignoring requests for services from an agent, introducing unacceptable delays for critical tasks, not executing the agent code at all, or terminating the agent without notification.

- *Other-to-platform*, in which external entities execute the conventional denial-of-service attacks, aimed at the underlying operating system or communication protocols.

3.10.1.3 Unauthorized access

An unauthorized access attack occurs when an entity accesses another entity without authorization. This attack can occur in the following threat categories:

- *Agent-to-agent*, in which an agent attempts to directly access parts of other agents. For example, an agent can invoke the public methods of another agent (e.g. attempt a buffer overflow, or reset the agent to its initial state), or access and modify the agent's data or code. This is a serious threat because if the agent platform has weak or no control mechanisms, modification of an agent's code can change a trusted agent into a malicious one.

- *Agent-to-platform*, in which an agent accesses the agent platform and its services without proper authorization. Such attacks are the result of inadequate access controls and agent authentication.

- *Other-to-platform*, in which remote users, processes, and agents request services and resources of the agent platform to which they are not entitled.

3.10.1.4 Repudiation

Repudiation is applicable to the agent-to-agent threat category. It occurs when an agent has participated in a transaction or communication, but claims that the transaction or communication did not take place. For example, a malicious agent may request some service from another agent, but later deny that it requested the service.

3.10.1.5 Eavesdropping

This attack is applicable to the platform-to-agent threat category. It occurs when an agent platform secretly monitors the communication between agents on the platform, monitors the instructions executed by agents, as well as all unencrypted data brought by the agent to the platform and the data generated by the agent on the platform. Even if agents do not expose any sensitive data, the platform can infer the meaning from the

types of services requested and from the identities of the agents with which it communicates.

3.10.1.6 Alteration of agents

This attack is applicable in the platform-to-agent threat category, and occurs when an agent platform alters the code, state, or data of an agent. Alteration can also occur to agent communications, where the platform attempts to alter the contents of messages passed between agents.

3.10.1.7 Copy-and-replay

The copy-and-replay attack is applicable to the other-to-agent platform threat category. It occurs when an interceptor attempts to intercept an agent or an agent message in transit in order to copy the agent, or agent message, and clone it or retransmit it. This may disrupt the synchronization or integrity of the agent framework.

3.10.2 Countermeasures

In this subsection, only an outline of the different countermeasures available against the above threats is provided. The countermeasures are discussed in detail in Jansen (1999) and in Jansen and Karygiannis (1999).

3.10.2.1 Countermeasures against platform threats

Countermeasures against platform threats include both conventional mechanisms employed in distributed applications and also those specifically developed for protecting mobile agent platforms.

The conventional mechanisms employed in distributed applications that can be applied to protecting mobile agent platforms include:

- Mechanisms for isolating processes from one another and from the control process.
- Mechanisms for controlling access to computational resources.
- Mechanisms for auditing security-relevant events occurring at the agent platform.
- Cryptographic methods for enciphering information exchanges and for the identification and authentication of users, agents and platforms.

Techniques developed more recently aimed at mobile code, and which are applicable to mobile agent security, include the following:

- Using interpreted script or programming language to develop agents.
- Limiting the capabilities of agent languages so that they are considered "safe".
- Using digital signatures and other information to indicate the authenticity of agents.
- Using various techniques to restrict an agent's capabilities, including constraining the resources that an agent may use (e.g. by constricting the lifetime and storage of an agent), controlling access to services (e.g. network destinations), and making capabilities location-dependent.

3.10.2.2 Countermeasures against agent threats

Countermeasures against agent threats tend to concentrate on detecting, rather than on preventing malicious behaviour. This is due to the fact that an agent is completely susceptible to an agent platform and cannot prevent malicious behaviour from occurring, but may be able to detect it.

If it is assumed that an agent trusts its home platform to provide the required support services and will not subvert the agent's activities, then the platform can be allowed to

apply conventional security techniques as countermeasures on behalf of the agent. These counter measures include the following:

- Issuing users and agent platforms public key certificates for authentication.
- Conveying information, such as agents and their messages, securely among agent platforms.
- Detecting and ignoring replay attacks aimed against agent platforms.
- Enabling an agent to audit platform services and other security-related events for post-processing analysis and detection.

Besides these conventional techniques, there are other approaches that have also been proposed. These include the following:

- Subjecting agents to state appraisal as a complement to signed code. This is to ensure that an agent has not been subverted as a result of alteration of its state information.
- Proof carrying code, which requires agents to carry proof of safety properties of its code. For example, the code producer (e.g. the agent's author) must formally prove that the program possesses safety properties that have been previously stipulated by the code consumer (e.g. the security policy of the agent platform).
- Requiring agents to maintain a record of the platforms visited by the agent.
- Requiring agent platforms to maintain execution traces of an agent's code.
- Enabling agents to execute encrypted functions safely at an agent platform.

3.11 Conclusion

The purpose of this chapter was to provide theoretical background information on mobile agents. First, the mobile agent paradigm was compared to other network computing paradigms. Then, the basic elements of a mobile agent system were discussed, together with the basic mobile agent behaviour. This chapter also discussed the potential advantages and disadvantages of applying mobile agents to intrusion detection. Security issues related to the use of mobile agents have also been discussed. The next chapter provides a brief overview of the human immune system.

CHAPTER 4

AN OVERVIEW OF THE HUMAN IMMUNE SYSTEM

4.1 Introduction

The purpose of the human immune system is to protect the body from harmful invaders, such as bacteria, viruses, fungi and parasites. In this chapter, the human immune system is briefly discussed. This will be necessary to understand how its concepts have been applied to the IDS proposed in the next chapter. The chapter is organized as follows. In Section 4.2, a short overview of the human immune system is presented. This is followed by Section 4.3, which discusses some of the immune system's properties that enable it to effectively detect intrusions. This chapter is concluded in Section 4.4.

4.2 An overview of the human immune system

The overview presented in this section is largely incomplete and simplified, with many important aspects of the immune system not discussed. Only enough information is provided to understand how its concepts have been applied to the IDS proposed in the next chapter. This overview is based on Davies (1997), De Castro and Von Zuben (1999), as well as the overviews of the immune system presented by Forrest, Hofmeyr and Somayaji (1997); Somayaji, Hofmeyr and Forrest (1997); and Kim and Bentley (1999).

The human immune system is based on the concept of distinguishing molecules and cells of the body, called "self", from foreign ones, called "non-self", and elimination of the latter. Its function is implemented through the interactions between a large number of different types of cells rather than by one particular organ.

The architecture of the human immune system is multi-layered; its protection layers can be divided into:

- *Physical barriers.* The skin is an example of a physical barrier and forms the outermost barrier of protection. Other physical barriers include the secretion of oils or mucus in the respiratory and digestive tracts, as well as coughing and sneezing.

- *Physiologic barriers*. The body's temperature and pH present unfavourable life conditions for some invaders. In addition, some body fluids, such as saliva, tears and stomach acids, contain destructive enzymes that can kill certain bacteria.

- *The barrier formed by the innate and the adaptive immune system*. These systems eliminate pathogens that were successful against the physical and physiologic barriers and have entered the body. The innate immune system mainly consists of circulating scavenger cells (such as macrophages) that ingest extracellular molecules and materials. The adaptive immune system is responsible for immunity that is adaptively acquired during the lifetime of the organism. The rest of this section focuses on the adaptive immune system because of its ability to adapt to detect many different and previously unseen invaders.

The adaptive immune system can be viewed as a distributed intrusion detection system in the body. The organs of the adaptive immune system, called lymphoid organs, are positioned throughout the body (e.g. tonsils, thymus, and bone marrow). The lymphoid organs are responsible for the production, growth, development, and storage of lymphocytes, which are also known as white blood cells.

White blood cells play a major role in the adaptive immune system. They function as small, disposable and independent intrusion detectors that circulate through the body in the blood and lymph systems. Each white blood cell is specialized to ignore self-cells and bind to a small number of structurally related non-self cells. Therefore, they can be considered as negative detectors, because they detect non-self patterns and ignore self patterns.

Detection and binding to non-self cells is accomplished through special receptors on white blood cells. Binding of a non-self cell to a white blood cell occurs when molecular bonds are formed between a non-self cell and the receptors on the white blood cell. The structure of the receptors is such that they will bind to a particular peptide (a sequence of amino acids, which make up proteins). As different types of cells contain different proteins, the receptors allow a white blood cell to recognize specific non-self cells and bind to them. After binding, many events still take place, usually resulting in scavenger cells (macrophages) eliminating the antigen.

The ability of the human immune system to detect a large variety of pathogens is partly due to the process through which the binding regions, or receptors, of lymphocytes are created. The receptors are created by a pseudo-random genetic process, which generates a large variety of receptors. This process may, however, lead to the creation of receptors that will bind to self-cells. To prevent binding to self-cells, the lymphocytes must first mature in the thymus before they are released to the rest of the body. There are several maturation stages, one of which is a process called *negative selection*. During the process of negative selection, any lymphocyte that binds to a self-cell circulating in the thymus is destroyed. Those lymphocytes that do not bind to self-cells leave the thymus and become part of the active immune system.

4.3 Properties that enable the immune system to effectively combat intrusions

Based on a study of the human immune system, Somayaji, Hofmeyr and Forrest (1997), and Forrest, Hofmeyr and Somayaji (1997) have presented many of its properties that can serve as design principles of a computer immune system. The human immune system has evolved many important properties that enable it to effectively combat intrusions. The properties relevant to the proposed IDS are discussed below.

1. *Distributability.* The immune system is highly distributed. No central co-ordination takes place; infections can be locally recognized by lymphocytes. Distributability greatly enhances the system's robustness.

2. *Diversity.* The immune system of each individual in a population is unique. This ensures that not all individuals will be vulnerable to the same pathogen to the same degree, thereby enhancing the survival of the population as a whole.

3. *Disposability.* Any cell of the immune system can be replaced. Therefore, there is no single component that is essential to the system's function.

4. *Autonomy*. There is no need for outside management or maintenance in the immune system; pathogens are autonomously classified and eliminated.

5. *Adaptability.* The system can adapt by learning to detect new pathogens. At the same time, it is also able to recognize previously seen pathogens through immune memory.

6. *Anomaly detection*. The immune system is said to perform anomaly detection because it is able to detect pathogens that it has not encountered before.

7. *Dynamically changing coverage*. The immune system is unable to maintain a set of detectors large enough to cover the space of all pathogens. Therefore, at any time, only a random sample of detectors circulates throughout the body. This sample of detectors is constantly changing through cell death and reproduction.

8. *Identity via behaviour*. Peptides, or protein fragments, serve as indicators of behaviour through which identity is verified.

9. *Detection is imperfect.* Not every pathogen is matched exactly by a pre-existing detector. This increases the system's flexibility with which it can allocate resources. For example, although a less specific lymphocyte will be less efficient at detecting a particular pathogen, it can detect a greater variety of pathogens.

These properties of the human immune system not only enable it to effectively detect and eliminate intrusions, but they also make the system fault-tolerant. The system can remain functional even when many of its components have failed.

4.4 Conclusion

The role of the immune system in the body is analogous to that of a computer security system in computing (Forrest, Hofmeyr & Somayaji, 1997). In the case of a computer security system, "self" may be defined as any normal activity and "non-self" may be defined as any form of abnormal activity, such as an unauthorized user, virus, worm, or Trojan horse. Although many differences exist between living organisms and computer systems, the similarities could point the way to improved computer security.

If the properties of the immune system are applied to an IDS, together with the capabilities and advantages of mobile agents, then the fault-tolerance of the IDS can be improved. Such an IDS can also remain operational even when most of its components have been attacked and disabled. A possible IDS design based on mobile agents and which applies some of the properties of the immune system is described in the next chapter.

CHAPTER 5

AN INTRUSION DETECTION SYSTEM BASED ON CONCEPTS OF THE HUMAN IMMUNE SYSTEM AND MOBILE AGENTS

5.1 Introduction

In the previous chapter, some of the main properties of the human immune system have been briefly discussed and included the following:

- The system uses anomaly detection to detect intrusions.
- Recognition of intrusions is based on distinguishing "self" from "non-self".
- Lymphocytes, or white blood cells, play an important role in the human immune system. They can be viewed as small, independent, and disposable intrusion detectors that circulate in their domain.
- The detectors are stored in and released by lymphoid organs.

Several approaches for building computer security architectures that incorporate principles of the human immune system have been proposed by Somayaji, Hofmeyr and Forrest (1997). One approach suggested was to implement the adaptive immune system layer by kernel-assisted lymphocyte processes, or mobile agents, that can migrate between computers. With help from the kernel, the lymphocyte processes are able to query other processes to determine if they are functioning normally. Nevertheless, the authors did not provide further details about this particular suggested approach. We used this suggestion in our earlier work (Zielinski & Venter, 2004) and proposed an IDS based on the above properties of the immune system. In this chapter, we discuss our proposed IDS in greater detail.

This chapter is organized as follows. Section 5.2 defines "self" in the context of an IDS and explains how it is used by the IDS. In Section 5.3, mobile agents are discussed, which use the definition of "self" to act as independent intrusion detectors of the proposed IDS. Section 5.4 discusses lymphoid hosts, which are computers that act as the lymphoid organs of the immune system and are responsible for creating and releasing mobile agents. This is followed by Section 5.5, which contains a discussion of the two stages that comprise the operation of the IDS: the training stage and the anomaly detection stage. A discussion of how concepts of the immune system have been applied in the IDS is presented in Section 5.6. The chapter is concluded in Section 5.7.

5.2 A definition of "self"

Before the IDS can detect intrusions, we must define what will be considered as an intrusion. In the immune system, an intrusion is considered to be any cell or molecule that is foreign to the body; intrusions are recognized by distinguishing molecules and cells of the body, called "self", from foreign ones, called "non-self". In a similar way, we define the terms "self" and "non-self" in the context of the proposed IDS and use the definitions as a basis for recognizing intrusions.

In the proposed IDS, "*self*" is defined as the normal behaviour of privileged processes. Behaviour of a privileged process that significantly deviates from its normal, or "self", behaviour is termed "*non-self*" and is considered as a possible intrusion.

5.2.1 Defining normal behaviour

As a program executes, it requires certain services from the operating system, such as I/O or resource allocation. A process requests services from the operating system through system calls, which can be generated directly or indirectly (e.g. by calling a run-time routine that makes the system call) (Silberschatz & Galvin, 1999). Therefore, every process implicitly specifies a set of system call sequences that it can produce. The ordering of system calls in the set of the possible execution paths through a program

determines these sequences. Some subset of these sequences will be determined during the normal execution of a program. Although the theoretical sets of system call sequences for complex programs will be large, the local (short range) ordering of system calls appears to be remarkably consistent (Forrest et al., 1996).

Normal behaviour of a program is defined to be the collection of short sequences of system calls made by the program during its normal execution (i.e. when no intrusions occur). This definition has been introduced in Forrest et al. (1996) and is used by the proposed IDS. The system calls are observed and recorded during the training stage of the IDS, as described in Section 5.5.1.

Preliminary experiments performed by Forrest, Hofmeyr and Somayaji (1997) on a limited set of intrusions and other anomalous behaviour show that short sequences of system calls (e.g. with a length of 6) provide a compact signature for self that distinguishes normal from abnormal behaviour.

The IDS is restricted to monitoring privileged processes (i.e. processes with root / supervisor access) for several reasons. Privileged processes are allowed to bypass the security mechanism of the kernel in order to accomplish their tasks. They are also trusted not to compromise the security of the system. However, due to possible errors, privileged programs may have vulnerabilities, which can be exploited by attackers. Therefore, privileged processes are considered more dangerous than user processes because they have greater access to the computer system. In addition, a natural boundary with respect to external probes and intrusions is created by root processes, especially those that listen to a particular port. We have also decided not to monitor user behaviour because the "normal" behaviour of processes is far more limited and stable than the "normal" behaviour of users (Forrest et al., 1996; Ko, Fink & Levitt, 1994). However, users are monitored indirectly through the monitoring of any privileged processes created by them. This leads to a simpler definition of normal behaviour and could result in fewer false positives observed during the operation of the IDS.

The above definition of normal behaviour ignores certain aspects of process behaviour. For example, the parameter values passed to system calls and the instruction sequences between system calls are not considered. At this stage, this definition is sufficient for

our purpose, as our main goal is to demonstrate how an IDS can be designed using mobile agents and certain aspects of the immune system. If, in future, it will be necessary to include other aspects of process behaviour, then only the definition of normal behaviour will need to change, together with the mechanism used to distinguish between normal and abnormal behaviour.

5.2.2 Storing normal behaviour

A separate database, called a *self-database*, of normal behaviour for each desired program is created by examining the sequences of system calls made during the normal execution of a program. This occurs during the training stage of the IDS's operation, and is described in Section 5.5.1. The database is specific to a particular architecture, software version and configuration, local administrative policies, and usage patterns. Since there is a large variability in how individual systems are currently configured and used, the individual databases provide a unique definition of self for most systems (Forrest, Hofmeyr & Somayaji, 1997). The different databases defining self are stored by several hosts in the network, called *lymphoid hosts*, which are discussed in Section 5.4.

5.3 Mobile agents

The function of detecting intrusions is divided and placed within mobile agents, which travel from host to host in the network. Mobile agents form the data analysis component of the IDS and data is analysed at the same hosts where it is collected. Since the number and the locations of the data analysis components are dependent on the current number and the locations of the mobile agents in the computer network, the IDS is considered to be distributed.

The proposed IDS has one type of mobile agent for each type of privileged program that it can monitor. Each mobile agent can detect anomalies in the behaviour of a particular type of privileged program. Therefore, mobile agents do not actually detect intrusions, but rather detect anomalies in the behaviour of an executing program. The anomalies detected by an agent may indeed be signs of intrusions (i.e. a true positive). However, it

is also possible that an agent detects a behaviour anomaly that is not a sign of an intrusion. In such cases, the anomaly represents program behaviour that is legitimate, but which is treated as anomalous by the agent because this behaviour has not been recorded as part of the normal behaviour of the program (i.e. a false positive). A detailed description of the way in which the IDS uses mobile agents to detect intrusions is presented in Section 5.5.

The choice to restrict a mobile agent to monitor only a specific type of executing program has been made for two reasons. Firstly, by allowing an agent to recognize abnormal behaviour of only a specific program, the agent is kept small because it needs to memorize the normal behaviour of only that specific program. Secondly, this specialization resembles the approach found in the immune system, where each white blood cell is specialized to bind to only a limited set of structurally related pathogens (intruders). Similarly, a mobile agent of the IDS is specialized to detect anomalies in the behaviour of only a particular type of privileged process.

At any time, the types of mobile agents that are present in the network determine the types of privileged programs that can be monitored. The desired rate of intrusion detection (i.e. how quickly an intrusion should be detected) is determined by how many mobile agents of each type are present in the network.

5.4 Lymphoid hosts

As explained in Chapter 4, the lymphoid organs of the adaptive immune system are positioned throughout the body and store white blood cells. White blood cells function as small, disposable, and independent intrusion detectors that circulate through the body in the blood and lymph systems. Similarly, the IDS has several hosts, called *lymphoid hosts*, positioned at different places in a computer network. (The name "lymphoid host" has been chosen because these hosts play the role of the lymphoid organs of the body.) A lymphoid host possesses all the necessary capabilities required to send independent detectors (mobile agents) that can detect anomalies in the behaviour of privileged processes.

A lymphoid host has the following responsibilities:

1. It stores the self-database of each privileged program that the IDS can monitor.
2. It decides when and which mobile agents to create.
3. It receives intrusion alerts from mobile agents. It can also respond to the alerts in several ways.
4. It provides the user interface through which the system administrator can learn of intrusions, issue commands to and change the configuration of the IDS.

The function of creating mobile agents is static, because it is provided by the (static) lymphoid hosts. However, the function of intrusion detection is mobile because it is provided through mobile agents (although mobile agents report intrusions to a static component of the IDS). The mobility of the intrusion-detection function is one of the main differences between the proposed IDS and an IDS that is based on only static components.

Although all lymphoid hosts have the capability to create mobile agents for intrusion detection, at any time, only one lymphoid host, called the *primary lymphoid host*, will be assigned this responsibility. The remaining lymphoid hosts serve as backups in case the primary lymphoid host is disabled or is otherwise made unavailable (e.g. as a result of a communication link failure).

The backup lymphoid hosts are each assigned a number, from 1 to *n*, where *n* is the total number of backup lymphoid hosts. This is needed to decide which backup host will take over the functions of a failed primary lymphoid host. When the primary lymphoid host fails, the backup lymphoid host with the lowest number becomes the primary lymphoid host and takes over its functions.

To allow the backup lymphoid hosts to know when the primary lymphoid host is unavailable, the primary lymphoid host periodically sends the backup lymphoid hosts a special message. This message notifies the recipient that the sender is still operating. When a backup lymphoid host does not receive this message within a predetermined period of time, it can assume that the primary lymphoid host has failed, or that the link between the primary and the backup lymphoid host has failed, or that the message has

been lost. The backup lymphoid hosts then wait for another time period to receive this message. If, after this time, they do not receive the message, the lowest numbered backup lymphoid host becomes the primary lymphoid host.

Therefore, a backup lymphoid host has to wait for a period of time before it can decide that the primary lymphoid host has failed and that it should take over the failed host's function. During this period of time, there will be no assigned primary lymphoid host. However, this does not imply that the computer system, which the IDS is monitoring, can be exploited by an attacker during the period when there is no assigned primary lymphoid host - the created mobile agents are still active and can still detect signs of possible intrusions. That is, the lack of a primary lymphoid host does not affect the mobile agents' task of *intrusion-detection*.

The lack of a primary lymphoid host does, however, affect the mobile agents' *intrusion-reporting* task. That is, the mobile agents are no longer able to report possible intrusions to the primary lymphoid host and must report intrusions to another host. Whenever a mobile agent will want to report an intrusion to the original primary lymphoid host and will notice that it is not available, it will report the intrusion to the backup lymphoid host that now acts as the primary lymphoid host. Therefore, there should be a mechanism to ensure that mobile agents will know to which lymphoid host they should report intrusions when the primary lymphoid host is no longer available. For this purpose, we propose that the first task of the backup lymphoid host, when it becomes a primary lymphoid host, is to broadcast this fact to all the mobile agents. Although this approach will create additional network traffic, it is preferred over providing each mobile agent with a hard-coded list of all the lymphoid hosts available in the network.

5.5 Operation of the IDS

In the immune system, intrusions (in the form of pathogens) are detected by searching for abnormal or "non-self" peptide patterns (see Section 4.2). The proposed IDS employs anomaly detection, just as the immune system does, because intrusions are detected by searching for abnormal or "non-self" sequences of system calls generated during a program's execution.

The operation of the proposed IDS is based on the following three assumptions (Forrest et al., 1996):

1. There are no intrusions during the time when the normal behaviour of a program is determined. If an intrusion occurs while the normal behaviour of a program is determined, then that particular intrusion will be regarded as part of the normal behaviour of the program. Such an intrusion will not be noticed when the IDS will be monitoring that program.

2. The sequence of system calls executed by a program is locally consistent during its normal operation. The code of most programs is static and system calls occur at fixed locations within the code. Although the relative ordering of the invoked system calls will be changed by conditionals and function calls, no variation to short-range correlations will necessarily be introduced.

3. A short, unusual sequence of system calls will be executed when a vulnerability in a program is exploited. For example, if an intruder replaces code of a running program, it would likely execute a sequence of system calls that is not found in the normal database. It is also likely that a successful intruder will need to fork a new process in order to exploit the system. This fork should be detectable when it occurs. In addition, attacks will also be noticed when a program enters an unusual error state during an attempted intrusion, provided that the error condition executes a sequence of system calls that is not in the normal database. Therefore, to detect intrusions, the sequence of system calls executed must be sufficiently different from the ones observed during the training stage.

An IDS that employs anomaly detection must first create a profile of normal system behaviour, during which no intrusion takes place. Once the profile has been created, the IDS compares the current behaviour of the system with the behaviour recorded earlier. Therefore, there are two distinctive stages in the operation of the IDS: the training stage and the anomaly detection stage.

5.5.1 The training stage

Before the IDS can be used to detect intrusions, it must first be "trained", or learn the normal behaviour of the desired privileged programs. For this purpose, a self-database is created for each privileged program that should be monitored.

Several methods can be used to create a model of normal privileged behaviour, or a self-database (Somayaji & Forrest, 2000; Warrender, Forrest & Pearlmutter, 1999). These include decision trees, neural networks, hidden Markov models, and methods based on deterministic finite automata.

For the proposed IDS, the method chosen for creating a model of normal privileged behaviour (i.e. the self-database) is the same as that used by Somayaji and Forrest (2000). However, the IDS proposed in this dissertation differs significantly from the intrusion-detection method described by the above authors in that the intrusion-detection function is placed within mobile agents and not within static components.

The method was chosen within the following constraints. First, the model must be constructed in one pass over the data; both training and testing must be efficient enough to be performed in real-time. In other words, the method must be appropriate to use for on-line training and testing. Secondly, the method must be suitable for large alphabet sizes. In this case, the alphabet consists of all the different system calls (about 200 for UNIX systems). Lastly, the method must be able to create models that will be sensitive to common forms of intrusions.

The system calls made by a process can be obtained by using system call tracers. Such tools exist on most UNIX platforms, e.g. `strace` on Linux or `truss` on Solaris. These tools can provide a list of the system calls a program makes when it executes.

To create the self-database, a small fixed-size window is slid over the recorded sequence of system calls and the calls that precede the current call within the sliding window are recorded. A "pair" is formed by the current call and a call at a fixed preceding window position. The contents of a window of length x are represented

by x - 1 pairs. The collection of unique pairs over all the traces for a single program constitute the model of normal behaviour for that program. A window of size six can be used in the IDS, which is the standard default used by Somayaji and Forrest (2000). (Section 6.5.3 discusses how the size of the detector window affects the detection accuracy of the IDS and why a window of size six is appropriate).

The model of normal behaviour of a program can be formally represented as a 4-tuple (S, T, w, P) where:

S is the alphabet of possible system calls,

T is the trace; the sequence $t_0, t_1, \ldots, t_{r-1}, t_r \in S$,

w is the window size such that $2 \leq w \leq r$, and

P is the behaviour profile; the set of patterns associated with T and w.

The construction of the normal database is best illustrated with an example. Suppose a window of size 4 ($w = 4$) is used with the following recorded sequence of system calls:

s1, s2, s2, s3, s4, s5, s3, s1, s2, s5

The profile of normal behaviour is constructed by sliding the window across the sequence. For each call encountered, the calls that precede it at different positions within the window are recorded and numbered from 0 to w - 1, where 0 is the current system call. The windows obtained in this trace are shown in Table 5.1.

Current	Position 1	Position 2	Position 3
s1			
s2	s1		
s2	s2	s1	
s3	s2	s2	s1
s4	s3	s2	s2
s5	s4	s3	s2
s3	s5	s4	s3
s1	s3	s5	s4
s2	s1	s3	s5
s5	s2	s1	s3

Table 5.1 Uncompressed table representing normal behaviour

It is likely that a call that occurs more than once in a trace will be preceded by different calls in different contexts. Therefore, the explicit window representation is compressed by joining together lines with the same current value. This produces a compressed table representing normal behaviour, as shown in Table 5.2.

Current	Position 1	Position 2	Position 3
s1	s3	s5	s4
s2	s1, s2	s1, s3	s5
s3	s2, s5	s2, s4	s1, s3
s4	s3	s2	s2
s5	s4, s2	s3, s1	s2, s3

Table 5.2 Compressed table representing normal behaviour

This table can be stored in a fixed-size bit array. If $|\ S\ |$ is the size of the alphabet and w is the window size, then the complete model of normal behaviour for a single program can be stored in a bit array of size $|\ S\ | \times |\ S\ | \times (w - 1)$. For example, the implementation used by Somayaji and Forrest (2000), where $w = 6$ is used as the standard default, uses a 200×200 byte array, with masks to access the individual bits.

This shows that a single self-database (i.e. the model of normal behaviour of a single program) does not require large amounts of storage space. Therefore, the size of a single self-database should be small enough to make it practical to be placed in a mobile agent.

Once the normal behaviour of a program is known, it can be used to monitor the behaviour of the program whenever it runs. Monitoring is based on examining the behaviour (or the generated sequence of system calls) of an executing program and comparing it with the normal behaviour which has been learnt earlier. This process occurs during the anomaly detection stage, which is described in the next section.

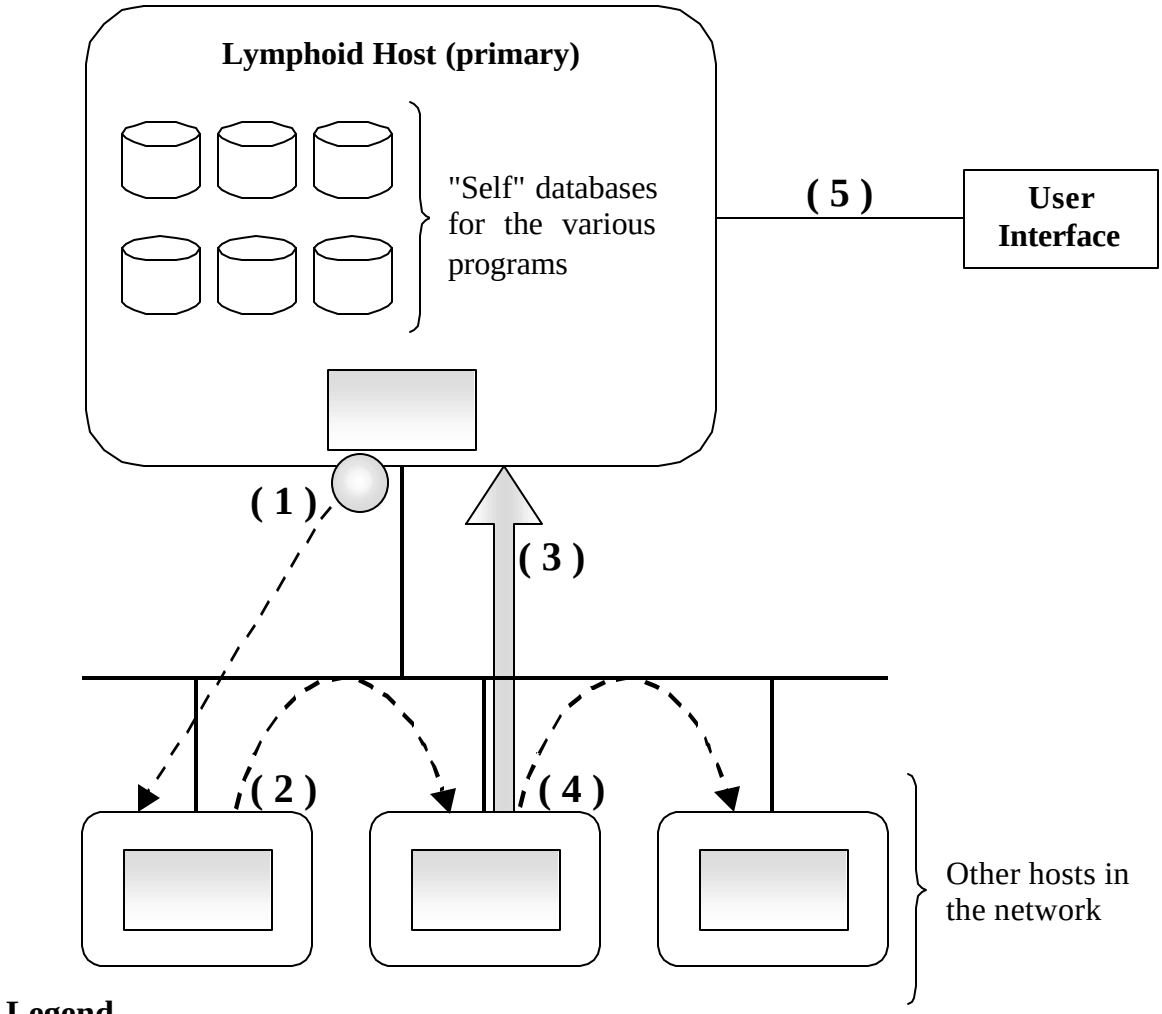
5.5.2 The anomaly detection stage

During the anomaly detection stage, the normal behaviour of a process is compared with its current behaviour. Significant deviations from the normal behaviour are classified as anomalous and as a sign of a possible intrusion.

The following steps describe how an intrusion would be detected by a single mobile agent during the anomaly detection stage:

1. The mobile agent is created and sent to the network.
2. The mobile agent traverses the network in search for a process to monitor.
3. The mobile agent monitors a privileged process.
4. The mobile agent reports an intrusion to the primary lymphoid host.

The operation of the IDS during the anomaly detection stage is shown in Figure 5.1.



Legend


Mobile agent platform


Mobile agent


"Self" database of a privileged program, which stores sequences of system calls that define the normal behaviour of that program


Path of mobile agent as it traverses the network

Computer network


Intrusion alert sent by mobile agent to primary lymphoid host

(1) A mobile agent is created by the primary lymphoid host and is sent to the network.
(2) The mobile agent traverses the network, searching for anomalies in the behaviour of a particular program.
(3) An anomaly has been detected by the mobile agent, which may be a sign of an intrusion. An intrusion alert is sent to the primary lymphoid host.
(4) The mobile agent continues to monitor other computers ...
(5) ... while the primary lymphoid host responds to the intrusion by taking some appropriate action (in this case, it notifies the system administrator).

Figure 5.1 Operation of the proposed IDS during the anomaly detection stage

5.5.2.1 Creation of a mobile agent

The configuration of the IDS determines the types and the number of mobile agents that may be created. The specific privileged programs that should be monitored at any time determine the types of mobile agents that may be created by a lymphoid host. The desired rate of intrusion detection (i.e. how quickly an intrusion should be detected) determines how many mobile agents of each type may be present in the network.

Mobile agents are created by the primary lymphoid host, according to the way in which the IDS is configured. The primary lymphoid host first creates a generic mobile agent; a mobile agent that is not yet specialized to monitor a specific privileged program. Specialization of a mobile agent occurs when the lymphoid host adds the self-database of a specific program to the agent.

5.5.2.2 Searching for a process to monitor

Once a mobile agent has been created and specialized, it is sent to the network to search for possible intrusions. The mobile agent moves from host to host by either using a list of hosts it should visit (provided by the lymphoid host), or by randomly choosing a host to visit. This choice is an implementation issue (discussed in Section 6.5.7) and does not affect the rest of the discussion in this section.

All the participating nodes, to which a mobile agent may travel, must have an agent platform installed. Since many agent systems operate over a wide range of hardware and software, this requirement is not as difficult to fulfil as it may first appear (i.e. it is possible for heterogeneous computers to have the same type of agent platform installed) (Jansen, 2002).

5.5.2.3 Monitoring of a privileged process

Once a mobile agent has reached a desired host, it determines whether the program, whose normal behaviour it stores, is executing. If the particular program is executing, then the mobile agent obtains system call sequences made by that program by using the same process as was used for learning the normal behaviour. As the program is

executing, the mobile agent compares the system calls made with those stored by the agent. The comparison process is performed as long as the program is executing. If the agent finds a significant deviation from the normal behaviour, it reports a possible intrusion to the lymphoid host. Otherwise, if the comparison does not show a significant deviation from the normal behaviour and the program has completed its execution, then the mobile agent searches for another process to monitor. The technique that is used to compare system calls is the one that has been described in Somayaji and Forrest (2000).

Any system call pair (the current call and a preceding call within the current window) not present in the normal profile is called a *mismatch*. Any individual mismatch could indicate an anomaly in the behaviour of the process (a true positive), or it could be a sequence that has not been included in the normal profile during the training phase (a false positive). The current system call is regarded as anomalous if there are any mismatches within its window.

A fixed-size circular array, called a *locality frame*, of size *n* is used to record the number of the past *n* system calls that were anomalous. Each element of the locality frame can store a value of 0 or 1. For the i^{th} system call made by a program, the $(i \text{ MOD } n)^{th}$ element of the locality frame is set a value of 0 or 1. A value of 0 indicates that no mismatch occurred in the current window, while a value of 1 indicates that at least one mismatch occurred in the current window. That is, each element of the locality frame does not record the number of mismatches in a window, but whether or not a mismatch has occurred. (The size of the locality frame is a user-set parameter and is not dependent on the size of the detector window.)

The sum of the elements of the locality frame is the total number of recent anomalies, or the number of the past *n* system calls that were rejected by the definition of normal behaviour. This value is called the *locality frame count* (LFC) and provides the anomaly signal for the IDS.

The value of the locality frame count does not depend upon the absolute trace length. Therefore, system calls can be monitored as they are made by a process, which allows this technique to be implemented in an "on-line" IDS. That is, it is not necessary to first

record the complete list of system calls made during the lifetime of a process before they are compared.

A user-defined number is selected as a threshold on the LFC value. (The selected number must be between 1 and the size of the locality frame). The threshold value controls the allowable deviation from the normal behaviour by indicating how many system call mismatches are allowed to occur before the mobile agent sends an intrusion alert. At any time, if the value of the LFC is below this threshold, no intrusions are reported. However, if the value of the LFC reaches or exceeds this threshold, it may indicate a possible intrusion and the agent sends an intrusion alert.

To explain the comparison process, the definition of normal behaviour defined in the previous example (Tables 5.1 and 5.2) will be used.

During the comparison process, the size of the detector window used must be the same as that which was used for learning the normal behaviour. In this example, the detector window has size 4.

Assume that, in this example, the size of the locality frame has been set to 10 and that the value of the threshold on the locality frame count has been set to 5. Therefore, the locality frame indicates the number of the past 10 system calls that were anomalous and at least 5 of the past 10 system calls must be anomalous for the mobile agent to report a possible intrusion. (The values for the size of the locality frame and the threshold for the locality frame count are user-defined; see Section 6.5.4 on how these values affect the detection accuracy of the IDS).

Let the sequence of system calls that is observed during the execution of a program be as follows (the ellipses indicate that the sequence is not a complete list of system calls made by the program).

s1, s2, s2, s3, s2, s4, s1, s5, s2, s1, s2, s5, s3, ...

Table 5.3 shows how the values of the elements of the locality frame change as the above system calls pass through the detector window. Each step is explained below the table.

Comparison steps	Examined system calls		Detector Window				Unexamined system calls		Elements of the Locality Frame									
			Position 3	Position 2	Position 1	Current			1	2	3	4	5	6	7	8	9	10
1						s1	s2	s2	0	-	-	-	-	-	-	-	-	-
2					s1	s2	s2	s3	0	0	-	-	-	-	-	-	-	-
3				s1	s2	s2	s3	s2	0	0	0	-	-	-	-	-	-	-
4			s1	s2	s2	s3	s2	s4	0	0	0	0	-	-	-	-	-	-
5		s1	s2	s2	s3	s2	s4	s1	0	0	0	0	1	-	-	-	-	-
6	s1	s2	s2	s3	s2	s4	s1	s5	0	0	0	0	1	1	-	-	-	-
7	s2	s2	s3	s2	s4	s1	s5	s2	0	0	0	0	1	1	1	-	-	-
8	s2	s3	s2	s4	s1	s5	s2	s1	0	0	0	0	1	1	1	1	-	-
9	s3	s2	s4	s1	s5	s2	s1	s2	0	0	0	0	1	1	1	1	1	-
10	s2	s4	s1	s5	s2	s1	s2	s5	0	0	0	0	1	1	1	1	1	1
11	s4	s1	s5	s2	s1	s2	s5	s3	1	0	0	0	1	1	1	1	1	1
12	s1	s5	s2	s1	s2	s5	s3	...	1	0	0	0	1	1	1	1	1	1
13	s5	s2	s1	s2	s5	s3	...	...	1	0	0	0	1	1	1	1	1	1

Table 5.3 Example of system call comparison

Step 1: The current system call is s1 and this is the first system call observed. Therefore the $(1 \text{ MOD } 10)^{th}$ element of the locality frame will be used to indicate whether or not this system call is anomalous. This entry occurs in the defined normal behaviour (first row of Table 5.1) and therefore is considered as normal. Therefore, element 1 of the locality frame is assigned a value of 0.

Step 2: The current system call is s2 and this is the second system call observed. The system call in position 1 is s1. Table 5.1 shows that system call s2 in the current position and system call s1 in position 1 is part of the normal behaviour of this program. Therefore, the second $((2 \text{ MOD } 10)^{th})$ element of the locality frame is set to 0, indicating that no mismatches occurred in this window.

Step 3 - 4: In a similar way, the next two system calls are compared with the normal behaviour. Elements 1 to 4 of the locality frame have a value of 0. The sum of these elements, or the locality frame count, is 0. This indicates that no mismatches occurred so far.

Step 5: The current system call is s2 and this is the fifth system call observed. Table 5.2 shows that the system call in position 1 should be either s1 or s2, but in this case s3 occurs in position 1. Therefore a mismatch has occurred. Mismatches also occur in positions 2 and 3 of this window. Therefore, the fifth element of the locality frame is set a value of 1, which indicates that at least one mismatch has occurred in this window.

Step 6: The current system call is s4. Table 5.2 shows that the allowable system call in position 1 should be s3. However, in this case, system call s2 occurs in position 1. Therefore a mismatch has occurred. A mismatch also occurs in position 2, where s3 occurs but where only s2 is considered as normal. Therefore, the sixth element of the locality frame is set a value of 1, indicating that at least one mismatch has occurred in this window.

Step 7 - 8: In a similar way, the next two system calls are considered as anomalous because at least one mismatch has occurred in each window. The elements of the locality frame are updated accordingly. The value of the LFC at the end of step 8 is 4.

Since the threshold set on this value is 5, the mobile agent still does not report a possible intrusion.

Step 9: Another system call is considered anomalous because a mismatch has occurred in this window. The ninth element of the locality frame is set to 1, and now the value of the LFC is 5. The threshold on the LFC value has been reached and the mobile agent sends an alert of a possible intrusion.

Step 10: Another system call is considered anomalous because a mismatch has occurred in this window. The tenth element of the locality frame is set to 1, and now the value of the LFC is 6. The mobile agent has already sent an intrusion alert in step 9. Whether or not the mobile agent sends another intrusion alert at this stage is an implementation issue.

Step 11: Another system call is considered anomalous; this system call is the eleventh system call observed. Therefore the $(11 \bmod 10 = 1)^{th}$ element of the locality frame is set to 1. The value of the LFC is now 7. This indicates that, of the last 10 system calls observed, 7 are considered not to be part of the normal behaviour of this program.

Step 12: The system call in the current position of the detector window is s5. Table 5.2, shows that the allowable system calls (or the system calls considered as part of normal behaviour) are: s4 or s2 in position 1, s3 or s1 in position 2, and s2 or s3 in position 3. This is exactly the case and therefore there are no mismatches with the twelfth system call. The second element of the locality frame is set to 0, but the value of the LFC is still 7.

Step 13: No mismatches occur with the thirteenth observed system call; the value of the third element of the locality frame is set to 0. The ellipses in the next unexamined system call indicate that the program has still not completed executing and more system calls can occur.

5.5.2.4 Reporting an intrusion to the lymphoid host

When a mobile agent has detected a possible intrusion it sends an intrusion alert to the primary lymphoid host and continues to monitor the process. When the lymphoid host receives this message, it will respond by taking an appropriate course of action, as determined by the way in which the IDS has been configured.

Several possible actions that the IDS can take to respond to an intrusion are listed below. The list is not complete; a more thorough study of appropriate intrusion response actions is beyond the scope of this research.

1. Do not respond to the intrusion and let the mobile agent continue to monitor the process. This action assumes that, if a privileged process has actually been used as a means of intrusion, then additional anomalous system call sequences will be created. The mobile agent that monitors this process will be able to notice them and alert the lymphoid host again. When the lymphoid host receives more than one alert from the mobile agent, it responds to the intrusion by taking some other action.

2. Alert the system administrator through the user interface provided by the lymphoid host. This action assumes that the administrator will investigate the intrusion and take an appropriate action.

3. Increase the rate of intrusion detection for the program type that caused the intrusion. This is done by allowing the lymphoid host to create additional mobile agents, whose type is the same as that of the reporting mobile agent. This may also be done by allowing the reporting agent to create additional copies of itself. This depends on whether or not the IDS has been configured to allow the mobile agents this capability. This action assumes that, if an intrusion has occurred through a particular program running on a particular host, then additional intrusions may occur (or have occurred) at other hosts on which this particular program runs. The number of agents that can be created in such situations should be set by the user before the operation of the IDS.

4. Increase the number of types of privileged programs that can be monitored. This is done by allowing the lymphoid host to create additional types of mobile agents that are present in the network. This action assumes that, if an intrusion has occurred through a particular type of privileged process, then additional intrusions may occur (or have occurred) at hosts with other types of privileged processes. The number and types of agents created in this case should be set by the user before the operation of the IDS.

5.6 Immune system concepts applied in the IDS

In Section 4.3, properties of the immune system that make the system fault-tolerant and enable it to effectively detect intrusions were presented. By applying concepts of the immune system to an IDS based on mobile agents, an IDS resulted that also possesses those same properties. The same properties that offered fault-tolerance to the immune system have also contributed to the fault-tolerance of the IDS.

1. *Distributability.* As in the immune system, there is no central controller in the IDS. Intrusion detection is distributed because this function resides within mobile agents that travel from host to host in the network.

2. *Diversity.* A separate database of normal behaviour for each desired program is created. The database is specific to a particular architecture, software version and configuration, local administrative policies as well as usage patterns. Since there is a large variability in how individual systems are currently configured and used, the individual databases will provide a unique definition of self for most systems (Forrest et al., 1996).

3. *Disposability.* Any mobile agent of the IDS can be replaced by creating a new one. Therefore, no single mobile agent is essential to detect intrusions.

4. *Autonomy.* Once a mobile agent has been created, it is not directly controlled by some other entity. For example, the agent itself decides when to migrate from one host to another, without being directly controlled by its home platform.

5. *Adaptability*. The system can adapt to detect new types of intrusions because any new behaviour of a program (i.e. behaviour that has not been recorded in the self-database of that program) will be regarded as intrusive. At the same time, previously seen intrusions will also be recognized because they too have not been recorded as part of the normal behaviour of that particular program.

6. *Anomaly detection*. The IDS uses anomaly detection because it is based on identifying deviations in a program's normal behaviour rather than on searching for known attack methods.

7. *Dynamically changing coverage*. In order not to overburden the computer system, not all hosts with an installed agent platform are monitored at the same time. Only those hosts that have a mobile agent running on an agent platform are monitored. Even when a mobile agent is monitoring a particular host, only the program type for which the mobile agent is specialized is being monitored. In addition, the types of programs that are monitored can be changed by changing the types of agents that are present in the network. Therefore, the coverage of programs that are monitored changes dynamically.

8. *Identity via behaviour*. The IDS identifies processes via their behaviour. "Self" processes are identified by the system call sequences that are stored in the self-database of that program. Intrusive, or "non-self", processes are identified by the lack of the generated system call sequences in the self-database of that program.

9. *Imperfect detection*. The IDS creates an intrusion alert whenever the behaviour of a process significantly deviates from the behaviour that has been recorded in the self-database of that program. Therefore, detection is imperfect because it is not known whether the behaviour deviation is a sign of an actual intrusion or a false alarm.

In conclusion, we can see that the use of mobile agents in the proposed IDS is analogous to the use of white blood cells in the human immune system. Mobile agents and white blood cells have several similarities (largely based on Foukia, Hulaas and Harms (2001)). Firstly, both mobile agents and white blood cells are autonomous. That is, once they have been created, they are not directly controlled by other entities (i.e.

other organs in the case of white blood cells or other computers in the case of mobile agents). Secondly, both continuously circulate through their domain: white blood cells continually move through the body in the blood, while mobile agents continually move from computer to computer through the network. Thirdly, both are specialized in detecting a particular intrusion. Each white blood cell is specialized to detect only a particular type of antigen (e.g. a specific kind of bacteria) and will not react to another kind of antigen. Similarly, mobile agents of the proposed IDS are specialized to detect anomalies in the behaviour of only a particular type of privileged program.

5.7 Conclusion

This chapter presented an IDS that is based on some of the main properties of the human immune system and which uses mobile agents to detect intrusions. The chapter discussed the main components of the IDS and how the IDS should function if it were implemented.

In the next chapter, the proposed IDS is theoretically evaluated by discussing how it satisfies the objective of this research, as well as the system's benefits and drawbacks. Some of the most significant issues that will need to be considered when implementing the proposed IDS are also discussed.

CHAPTER 6

THEORETICAL EVALUATION OF THE PROPOSED INTRUSION DETECTION SYSTEM

6.1 Introduction

In the previous chapter, an IDS that is based on the human immune system and which uses mobile agents to detect intrusions was proposed. In this chapter, the proposed IDS is theoretically evaluated by discussing how it satisfies the objective of this research, that is, how the IDS offers greater robustness and fault-tolerance than the present-day commercial IDSs based on hierarchical architectures. The benefits and the drawbacks of the proposed IDS are also discussed. This is followed by a discussion of some of the most significant issues that will need to be considered when implementing the IDS. This chapter is concluded in Section 6.6.

6.2 How does the proposed IDS solve the research problem?

The research problem, or the aim of the research, is to propose an IDS architecture that offers greater robustness and fault-tolerance than the current hierarchical architectures, which are used by nearly all present-day commercial IDSs. Therefore, to answer the question of how does the proposed IDS solve the research problem, we need to discuss how the proposed IDS provides greater robustness and fault-tolerance than an IDS based on a hierarchical architecture.

The main problem with a hierarchical IDS architecture (as discussed in Section 2.5.2) is that its tree structure introduces many single points of failure. This makes the system

vulnerable to direct attack because failure of such critical components (e.g. as a result of an attack on the IDS) may result in failure of the entire IDS. In addition, because such components are static in a network, they can be found relatively easily by a determined and knowledgeable attacker (when compared with the more difficult task of finding non-static components in a network). Once found, the components can be disabled by the attacker.

It is probably impossible to design an IDS that will be completely resistant to direct attack. However, an IDS can be designed such that it will be more robust and fault-tolerant and thereby more difficult to disable completely. By improving the robustness of the IDS, we reduce its vulnerability to direct attack. By increasing the fault-tolerance of the system, we provide it with the ability to tolerate failures of its key components. That is, the system can continue to function and detect intrusions even after most of its key components have been successfully attacked and disabled.

Our approach to improve the fault-tolerance of the IDS and to reduce the system's vulnerability to direct attack is to:

1. *Eliminate single points of failure in the IDS.* That is, there should be no component in the system that is essential to the system's function; every component should be replaceable. Such a system will be able to detect intrusions (although at a reduced efficiency) even when many of its components have failed. Therefore, the system will provide fault-tolerance.

2. *Make the critical components of the IDS non-static.* That is, the critical components of the IDS (i.e. the components that perform the actual function of intrusion detection) should be able to move from host to host in the network. This will make it more difficult for an attacker to know the exact location of each component at any time. In addition, it will be beneficial if these critical components were to have the ability to replicate. This will provide the IDS the ability to vary the number of the intrusion detection components. The implication of this is that an attacker would find it much more difficult to know the exact locations and the number of the critical components of the IDS. Therefore, the IDS will be less vulnerable to a direct attack

on its critical components, because they will be more difficult to locate than static components.

The rest of this section discusses how the above approach has been implemented and consequently improved the fault-tolerance and robustness of the IDS.

6.2.1 Elimination of single points of failure

The architecture of the proposed IDS has no aggregation nodes. There is also no root node that is responsible for the actual function of detecting intrusions. Instead, the function of intrusion detection is divided and placed within mobile agents, which travel from host to host in the network. Since the collection of mobile agents in the system provides the intrusion-detection function, the mobile agents are considered to be the critical components of the IDS (i.e. critical for the intrusion-detection function).

As explained in Section 4.2, the organs of the adaptive immune system (lymphoid organs) are positioned throughout the body and store white blood cells. Using the immune system analogy, the proposed IDS has two main types of components: lymphoid hosts and mobile agents. The lymphoid hosts play the role of the lymphoid organs, and the mobile agents play the role of the white blood cells. The lymphoid hosts are responsible for creating the mobile agents that detect intrusions. For this purpose, each lymphoid host requires a copy of the databases that define "self", or normal program behaviour. Therefore, copies of these databases are distributed throughout the network and are stored in every lymphoid host.

None of the main components of the IDS introduce a single point of failure. That is, the IDS can continue to function even when most of its components have been attacked or have failed.

When an intruder successfully attacks a lymphoid host (or otherwise makes the host unreachable), the mobile agents created by that host are still be able to function and carry out their task of intrusion detection. That is, a mobile agent does not rely on the presence of its lymphoid host to detect intrusions. This is possible because, once a

mobile agent has been created, it does not need to communicate with its home platform. (Once a mobile agent has been created, its only need for a lymphoid host is to send it reports of detected intrusions. However, it is also possible to report intrusions to some other host in the network.) When a lymphoid host is disabled, the backup lymphoid host takes over. This implies that the mobile agents in the network can report intrusions to a lymphoid host even if their home platform has been disabled. Even if all the lymphoid hosts have been successfully disabled, the mobile agents that remain in the network survive and can still detect intrusions and multiply, while intrusions can be reported to some other host at which the system administrator can be alerted.

It is also possible to increase the number of mobile agents in the IDS even if all the lymphoid hosts have been successfully disabled. This is possible through agent replication; an agent can replicate on any host where the agent platform is installed, without the assistance of its home platform. Therefore, the lymphoid hosts are not considered as single points of failure because intrusions can be detected by mobile agents even if all the lymphoid hosts have failed.

Mobile agents also do not introduce a single point of failure. As in the immune system, the individual intrusion detectors of the IDS (i.e. the mobile agents) are disposable. Any mobile agent of the IDS can be replaced by creating a new one. New mobile agents can be created either by a lymphoid host or through agent replication. Therefore, even if an attacker disables many mobile agents, intrusions can still be detected because new mobile agents can be quickly created by the system. Therefore, no single mobile agent is essential to detect intrusions.

Therefore, there are no single points of failure in the proposed IDS because no single component is essential for the system to detect intrusions. That is, the system can continue to function even when most of its components have failed; the IDS is fault-tolerant.

6.2.2 Making the critical components of the IDS non-static

The proposed IDS uses mobile agents to detect intrusions. Mobile agents, by definition, have the ability to migrate from one host to another in the network. Mobility of these critical components makes it more difficult for an attacker to disable them, thereby reducing the system's vulnerability to direct attack.

For the operation of the whole IDS to be disabled, an attacker would need to successfully disable all the lymphoid hosts and all the mobile agents in the network. When all the mobile agents and all the lymphoid hosts have been disabled, the IDS cannot detect intrusions and no new agents can be created either by the lymphoid hosts themselves or through agent replication.

The lymphoid hosts are static and therefore can be found and disabled by a determined and knowledgeable attacker. However, the mobile agents already in the network are more difficult to disable. To disable the mobile agents, the intruder would first need to find them, but since they are not static and their number can vary, this task is more difficult than for static components (such as the lymphoid hosts). The intruder could also try to attack the agent platforms themselves in an attempt to destroy the mobile agents. However, as every host that we wish to monitor will have an agent platform installed, this task is not trivial if there are many hosts.

Therefore, the proposed IDS is less vulnerable to direct attack. This results from the fact that the critical components of the IDS are more difficult to find and consequently are more difficult to disable than in a hierarchical IDS architecture.

6.3 Benefits of the proposed IDS

The proposed IDS offers a number of useful advantages for intrusion detection systems over an IDS based on a hierarchical architecture that uses only static components. In this section, the most significant advantages are discussed; this list is not necessarily exhaustive.

6.3.1 Greater robustness and fault-tolerance

The main benefit of the proposed system is that it offers greater robustness and fault-tolerance than hierarchical IDS architectures. The system has no single point of failure and can continue to operate even when most of its components have failed. This advantage has been described in detail in Section 6.2.

6.3.2 Reduction in network traffic

In a hierarchical IDS architecture, the nodes at the bottom of the hierarchy collect event data. This data is then passed to higher internal nodes until it reaches the root node. The internal nodes perform data aggregation, abstraction, and reduction. The root node is responsible, inter alia, for the actual function of determining whether an intrusion has occurred. The need to move large volumes of data among the nodes of the IDS produces a large amount of network traffic, especially if there are many nodes that comprise the architecture of the IDS.

The proposed IDS uses mobile agents for intrusion detection. The mobile agents move to the hosts at which event data is collected and process the data locally at that host. Therefore, there is no need to move the event data across the network. In other words, the computation is moved to the data, rather than moving the data to the computation. Although the movement of mobile agents will, of course, create network traffic, the amount of traffic generated will be lower (provided that the amount of event data collected at a host is larger than the size of a mobile agent that is assigned to process the data).

6.3.3 Dynamic adaptation

The proposed IDS can adapt its monitoring strategy. That is, if there is a low risk of an intrusion, then there may be a relatively small number of active mobile agents in the network. In such situations, the IDS imposes a small overhead on the computer system because there is a small number of active intrusion detectors (mobile agents) in the network. When the risk of an intrusion is greater, for example when intrusions have

occurred recently or when an intrusion is occurring, then the number of active mobile agents in the network can be increased. This will improve the intrusion detection rate (i.e. how quickly intrusions can be detected) because there are more intrusion detectors that search for intrusions in the network. Although the overhead imposed on the IDS during such situations is greater, it is a small cost to pay for the ability to quickly detect intrusions.

Therefore, the advantage of the ability to adapt to both favourable and unfavourable situations is that the overhead imposed on the computer system by the IDS can be adjusted according to the perceived intrusion risk.

6.3.4 Ability to detect attacks that a network-based IDS will miss

A network-based IDS can only analyse network traffic; it is unable to monitor the behaviour of executing programs. The proposed IDS can monitor the execution of specific privileged programs to determine whether some of their vulnerabilities have been exploited. Therefore, the proposed IDS can detect intrusions that a network-based IDS cannot.

6.3.5 Ability to detect unknown attacks

The proposed IDS is based on anomaly detection, which allows it to detect new types of attacks that have not been seen before. Therefore, unlike IDSs based on misuse detection, it does not need to be updated with new attack signatures when new attacks are devised.

6.3.6 Scalability

In the hierarchical IDS architecture described in Section 2.5.2, the root node performs the actual function of determining whether an intrusion has occurred. As more computers are added to computer networks, the computational load of the root node increases because it must process greater volumes of information.

To provide scalability in the proposed IDS, the total computational load of the intrusion-detection function is not placed on any single component of the IDS, but is distributed by using mobile agents. Each mobile agent monitors hosts locally, without the need for a central controller (i.e. there is no central controller that detects intrusions).

6.4 Drawbacks of the proposed IDS

Although the proposed IDS provides greater robustness and fault-tolerance, it is by no means perfect. Below, several weaknesses of the IDS are discused; the list is not necessarily exhaustive.

6.4.1 Inability to detect some types of intrusions

In the proposed IDS, sequences of system calls of executing privileged programs have been chosen as that on which monitoring is done. By making this choice, monitoring of the IDS is restricted to processes only. Therefore, there are several classes of intrusions that will not be detected:

- Because the IDS is host-based, it cannot detect attacks that are detectable by using a network-based IDS. For example, the TearDrop attack, which involves sending fragmented IP packets that overlap (CERT, 1997) cannot be detected by the IDS because this attack is only detectable by analysing network traffic.

- Attacks involving a race condition will not be detected (Forrest et al., 1996). These types of intrusions typically involve stealing a resource (such as a file) created by a program running with root / supervisor access, before the program has had a chance to restrict access to the resource. If the root process does not detect an unusual error, then a normal set of system calls will be made, which will not be regarded as an intrusion.

- Another type of intrusion that will not be detected is the case of an intruder using another user's account (Forrest et al., 1996). The IDS does not make use of user

profiles (which store a record of a user's normal behaviour) and therefore this class of intrusions is not likely to be detectable.

6.4.2 Dependency on the host operating system

The proposed IDS monitors the behaviour of privileged programs by comparing the system calls generated during their execution with those observed during the training phase. Because system calls are used as the basis of detecting intrusions, the intrusion-detection function is dependent on a specific set of system call names. This set of system call names is dependent on a specific operating system. Therefore, the IDS is dependent on a specific operating system used by the monitored hosts. Therefore, the IDS can be used to monitor only those hosts that are using the operating system(s) that were selected during the training phase (unlike a network-based IDS that is operating-system-independent).

6.4.3 Agent platforms

An agent platform must be installed on every host that should be monitored by mobile agents. Since many agent systems operate over a wide range of hardware and software, this requirement is not as difficult to fulfil as it may first appear. However, it may increase the cost of ownership of the IDS, when compared with a network-based IDS (which requires fewer data collection components).

6.4.4 Mobile agent security issues

Different categories of security threats related to the use of mobile agents have been discussed in Section 3.10. It is important to ensure that the use of mobile agents does not introduce vulnerabilities in the IDS. Therefore, before the system is implemented, it should be determined what countermeasures should be used to resolve or reduce the different threats.

6.5 Issues that should be considered when implementing the proposed IDS

There are many issues that need to be taken into account when implementing the proposed IDS. This section discusses the most significant issues that should be considered.

6.5.1 Dynamically changing coverage

The intrusion-detection function is implemented in mobile agents, with each mobile agent specialized to monitor only a specific type of program. The execution of a particular program on a host is monitored only when a mobile agent specific to that program is located on that host.

The implication of this is that a host on which a privileged program executes is vulnerable if there is no agent monitoring the execution of that privileged program. If the proper agent is not present on the host when a privileged program executes, any abnormal behaviour of that program will not be noticed. Therefore, even though the IDS can detect anomalies in privileged programs, it is possible that an intrusion will be unnoticed if there is no agent to detect it on the host when the attack occurs.

To ensure that a host is not left vulnerable whenever a privileged process executes, there should be a mobile agent, which is specialized in detecting intrusions in that specific program, located at the host. Increasing the number of mobile agents of each type decreases the likelihood of an undetected intrusion resulting from a lack of the appropriate agent to detect the intrusion as it occurs. However, increasing the number of mobile agents present in the network also increases the computational load on the computer system. It also increases the amount of network traffic generated by migration of agents. Therefore, there is a trade-off between the ability to detect all intrusions and the overhead placed on the computer system by the IDS.

One way to reduce the possibility of not detecting an intrusion is to record the system calls made by a privileged process in a log. This log is created on the host as the

privileged program executes. When the appropriate mobile agent arrives at the host, it searches for possible intrusions by comparing the system calls stored in the log with the system calls stored by the agent. The benefit of this approach is that all intrusions that can be detected by the IDS will be detected. However, this approach does not perform "on-line" intrusion detection because there can be a long time period between an intrusion and its detection. Therefore, the drawback of this approach is that it does not allow intrusions to be detected as soon as they occur, in time to respond to the intrusion while the attack is still occurring. In addition, during the time between the occurrence of an intrusion and its detection by a mobile agent, the intruder may already have gained sufficient privileges in the host to modify the log and erase any signs of an intrusion. Therefore, before the system is implemented, it should be considered whether it will be acceptable to allow some hosts to be vulnerable or whether the IDS should use logs, which would allow the IDS to detect all intrusions, even if it would not detect them as soon as they occur.

Another way to address this problem is to modify the kernel of the operating system such that, whenever a privileged program starts executing, the operating system sends a request to the primary lymphoid host to send an appropriate mobile agent to monitor the execution of the privileged program. Although this approach will ensure that the proper agent is always present on a host when a privileged program executes, it also requires that the operating system know which host in the network acts as the primary lymphoid host.

6.5.2 Mobile agent language

The language that will be used to implement the mobile agents must be selected not only with regard to its capabilities, but also with regard to its implications on the IDS.

For example, Java-based mobile agents typically load their class files dynamically, as needed, from their home platform (Jansen et al., 1999). If the home platform is unavailable (i.e. the lymphoid host has failed), these class files must be provided by the local host, or they must be found and transferred from some other remote host. Therefore, such mobile agents are not completely independent of their home platforms.

In addition, class loading from a remote platform or the local host platform also raises several security concerns. For example, the class files may have been modified, which can alter the functionality of the agent.

Moreover, the runtime environments for mobile agents often slow down an IDS based on mobile agents. This is especially apparent if mobile agents are implemented in slow interpreted languages.

6.5.3 Detector window size

During both the training stage and the anomaly detection stage, a detector window of a fixed size is used to obtain sequences of system calls. Before we discuss how the size of the detector window affects the detection accuracy of the IDS, let us define the terms *foreign sequence* and *minimal foreign sequence*.

A *foreign sequence* is a (Tan & Maxion, 2002) "... foreign order of symbols, i.e. a sequence in which each individual symbol within the sequence is a member of the training set alphabet, but where the order of the symbols is one that does not exist in the set of sequences obtained from the training-set ... ". In other words, it is a sequence that does not occur in the trace(s) used to define normal behaviour. A *minimal foreign sequence* is a foreign sequence that does not contain smaller foreign sequences.

A foreign sequence is only visible if the size of the detector window is at least as large as the size of the foreign sequence (Tan & Maxion, 2002). Therefore, the size of the detector window must not be smaller than the size of the minimal foreign sequence. This stems from the fact that the minimum size of the detector window required to detect each minimal foreign sequence is equal to the size of the minimal foreign sequence itself.

Somayaji and Forrest (2000) used a default detector size of length six. This is because the length of the smallest minimal foreign sequence present in their test data was six, which is the minimal foreign sequence length in this case.

The proposed IDS will not detect anomalies when the size of the foreign sequence anomaly is greater than the size of a foreign sequence detectable by the IDS (i.e. the size of the detector window). Therefore, the size of the detector window affects the detection accuracy of the IDS. To select an appropriate detector window size, the size of the minimal foreign sequence should be determined for the particular environment in which the IDS will be used.

6.5.4 Locality frame count

In the proposed IDS, a fixed-size circular array, called a *locality frame*, is used to record the current anomalous system calls. A locality frame of size *n* stores the number of the past *n* system calls that were anomalous. This number, called the *locality frame count* (LFC), is the total number of recent anomalies (mismatches) and provides the anomaly signal for the IDS. (The size of the locality frame is a user-set parameter and is not dependent of the size of the detector window.)

In the proposed IDS, a number must be chosen as a threshold on the LFC value, below which traces are still considered as normal. (This number must be between 1 and the size of the locality frame.) The threshold value controls the allowable deviation from the normal behaviour by indicating how many system call mismatches are allowed to occur before an intrusion alert is raised. If, at any time, the LFC value is equal to or greater than the chosen threshold, an anomaly is recorded.

The selection of the threshold number affects the detection accuracy of the proposed IDS. When a low threshold value is used, anomalies (intrusions) are recorded more often, but also with a greater number of false positives. That is, a small number of mismatches are needed for the IDS to record an intrusion. On the other hand, when a high threshold value is used, the number of false positives decreases, but so does the number of recorded anomalies. That is, many mismatches need to occur before the IDS will record an intrusion. Therefore, proper selection of a threshold value on the LFC plays an important role in the operation of the IDS. To select an appropriate threshold value on the LFC, it should be determined what the acceptable number of false positives

is in the particular environment in which the IDS will be used (bearing in mind that the threshold value also affects the number of true positives recorded).

6.5.5 Control of the IDS

In a hierarchical IDS architecture, the root node not only performs the actual function of intrusion detection, but it also usually reports to an operator console at which an administrator can manually assess status and issue commands to the IDS.

In the proposed IDS, there is no root node. As described in Section 5.5.2, when a mobile agent detects an intrusion, it alerts a lymphoid host. For this reason, the lymphoid host provides the user interface through which the system administrator can learn of intrusions, issue commands and configure the IDS.

However, when all lymphoid hosts have been disabled, mobile agents can still detect intrusions, but they will not be able to report intrusions to a lymphoid host. In such situations, the system administrator will not be able to learn of intrusions through a lymphoid host. Therefore, before the IDS is implemented, a choice must be made as to which computer a mobile agent must report intrusions when all lymphoid hosts have been disabled. This will allow the system administrator to know when an intrusion has occurred, even when all the lymphoid hosts have been disabled.

6.5.6 Intrusions during the learning stage

The operation of the proposed IDS is based on several assumptions (see Section 5.5). One assumption is that no intrusions occur during the time when the normal behaviour of a program is recorded (i.e. during the learning stage). If an intrusion occurs while the normal behaviour of a program is recorded, then that particular intrusion will be regarded as part of the normal behaviour of that program. Therefore, that particular intrusion will not be noticed when the IDS will be monitoring the behaviour of that program. Therefore, before the IDS is used, ways in which to ensure that no intrusions occur during the learning stage need to be examined. For example, the environment could be carefully monitored during the learning stage to ensure that no intrusions

occur, or the learning stage could occur in an secure, isolated environment, where it certain that no intrusions will occur.

6.5.7 Selection of monitored hosts

In the proposed IDS, mobile agents move from one host to another to detect intrusions. The exact way in which a mobile agent selects a host to which it will migrate is left to the implementation of the system.

One possible way in which to select hosts is to provide the mobile agent with a list of hosts that it must monitor. This list can be provided by the entity that created the mobile agent (e.g. the lymphoid host). The mobile agent then monitors the listed hosts, one after the other. The implication of this choice is that the creating entity must know what hosts should be monitored. Therefore, when hosts are added or removed from the network, the creating entity must be notified of such changes.

A different strategy is to allow the mobile agents to choose hosts by themselves. The mobile agent can choose a host randomly and migrate to that host. Once the mobile agent finished monitoring that host, it chooses another host at random and migrates to it. Therefore, a mobile agent is free to monitor any host it chooses. However, in this strategy, there is no control over which hosts will be monitored. This may lead to undesirable situations where some host is monitored more often than needed, while another host is not monitored enough.

Other strategies can also be considered. The benefits and drawbacks of different strategies need to be considered before an appropriate strategy is chosen.

6.5.8 Agent destruction

Agent destruction is necessary to allow the IDS to control the number and the types of mobile agents that are active in the network at any time. However, a vulnerability is introduced in the IDS by providing a component of the IDS the ability to destroy the intrusion detectors (mobile agents). That is, an attacker can modify this component in

such a way that it will issue commands to destroy mobile agents, thereby disabling the intrusion-detection function of the IDS.

Below is a suggested approach to agent destruction. This approach is based on providing each mobile agent with a limited lifetime, thereby ensuring that a mobile agent will not be active indefinitely. This is similar to the immune system, where each white blood cell also has a limited lifetime.

Initial approach to agent destruction

When a mobile agent is created, it is given a certain amount of "energy". This can refer to the number of hosts the agent can visit, or the amount of resources (e.g. CPU time) it can consume. As the agent traverses the network and monitors processes, it decrements its energy by a certain amount. When the energy of the agent reaches zero, the agent requests additional energy from the primary lymphoid host. Upon receiving this request, the primary lymphoid host decides if the agent should remain active or if the agent should be terminated. If the agent is to remain active, the primary lymphoid host responds by sending a message that allows the agent to increase its energy level by a certain amount. On the other hand, if the agent is no longer needed, then the primary lymphoid host notifies the agent that it will not receive additional energy. In this case, the agent will be terminated on its current agent platform.

In situations where all the lymphoid hosts have been attacked and disabled, the mobile agents would not be able to request additional energy. However, when all the lymphoid hosts have been attacked and disabled, the mobile agents in the network should continue with their task to ensure that the intrusion-detection function is not disabled when the static components of the IDS have been successfully attacked. Therefore, whenever a mobile agent cannot contact any lymphoid host with a request for additional energy, the agent assumes that all lymphoid hosts have been disabled. In such situations, the agent ignores the fact that it has no more energy and continues to function.

The described approach to agent destruction has a significant flaw. The flaw is the ability of a lymphoid host to notify a mobile agent that it will not receive additional energy, causing the agent to be destroyed at its current platform. An attacker can exploit

this fact by successfully attacking and modifying the lymphoid hosts in such a way that they will always notify agents that they will not receive additional energy. This can eventually lead to a situation where all active agents are destroyed and no guarantee is given that new agents will be created by the lymphoid hosts (since they are under the attacker's control). Therefore, an attacker will be able to disable the IDS without the need to find and destroy the mobile agents.

Modification of the initial approach to mitigate the described vulnerability

To eliminate a static component that provides additional energy to mobile agents, the IDS can have several mobile agents whose only purpose is to serve as energy-providers to the other mobile agents. This approach will make it more difficult for an attacker to disable mobile agents through the modification of the energy-providing component of the IDS because this component is mobile. Mobility makes this component more difficult to find by the attacker. Therefore, the vulnerability that the energy-providing component will be modified is reduced by reducing the possibility that this component can be found. However, this vulnerability is not eliminated because the energy-providing mobile agents are not immune to modification by an attacker.

Mobility also makes the energy-providing component more difficult to find by the other mobile agents in the network when they request energy. Either the energy-providing mobile agents need to be aware of the locations of the intrusion-detection agents in the network, or the intrusion-detection agents need to be aware of the location of the energy-providing component. Either way introduces additional overhead and complexity in the system.

Why is this aspect of the IDS considered as an implementation issue?

This aspect of the proposed IDS is probably best left to a specific implementation. The reason for this choice is that this vulnerability cannot be avoided. That is, although it is necessary to provide the IDS with the functionality of agent destruction (to control the number and the types of agents), no guarantee can be given that an attacker will not exploit this function. When this aspect is left to a specific implementation, it allows the implementer to address this vulnerability in a way that satisfies the needs of the

implementation. For example, when using the above approach, it is much more difficult to modify a lymphoid host rather than to disable it. Therefore, if the implementation requires that the IDS impose as little overhead as possible, then only the initial approach will be sufficient, at the risk that an attacker can modify the lymphoid hosts to disable the mobile agents. On the other hand, if modification of the lymphoid hosts is considered to pose a great risk to the IDS, then the implementer might choose to implement the modified approach, or choose to implement an entirely different approach.

6.6 Conclusion

In this chapter, the proposed IDS has been theoretically evaluated by discussing how it satisfies the objective of this research. The proposed IDS solved the research problem by eliminating single points of failure in the IDS, and also by making critical components of the IDS non-static. The possible benefits and drawbacks of the proposed IDS have also been discussed. This chapter also considered some of the system's unsolved aspects as well as the most significant issues that will need to be considered when implementing the IDS.

CHAPTER 7

CONCLUSIONS AND RECOMMENDATIONS

The purpose of this dissertation was to discuss how mobile agent technology can be applied in an IDS, and consequently increase the IDS's fault-tolerance. The human immune system was used as a model for the IDS. Such an IDS was proposed and theoretically described in terms of its main components and how the system functions. The resulting similarities between the human immune system and the proposed IDS have also been discussed.

Only certain properties of the human immune system have been applied in the proposed IDS. Many properties that are of importance to the human immune system have not been applied. That is, some aspects of the immune system, such as the various genetic controls used in the immune system, are not appropriate for an IDS. Other aspects of the immune system, such as random generation of receptors and negative selection, would unnecessarily complicate the proposed IDS. For example, random generation of receptors would require that the mobile agents be given a random sequence of system calls to search for (as opposed to be given a self-database and the ability to determine if the system call sequences are part of the self-database or not). Through the process of negative-selection, those mobile agents that have a sequence of system calls that is considered as part of the normal behaviour of a program, would be eliminated. Those agents that would not be eliminated, would be released and would search for their specific sequence of system calls. The number of possible sequences of system calls that are not part of the normal behaviour of a program far exceeds the number of those sequences that are part of the normal behaviour of a program. Therefore, such an IDS would need to create and manage a larger number of mobile agents than the proposed IDS. (This assumption would still be true even if each mobile agent would carry several sequences of system calls that are not considered as part of the normal behaviour of a program.)

First, background information to the solution was provided. This included Chapters 2, 3, and 4, where intrusion detection systems, mobile agents, and the human immune system were discussed, respectively. Chapter 2 also described why an IDS based on a hierarchical architecture has many single points of failure and is thereby not considered fault-tolerant (see Section 2.5.2). In Chapter 5, an IDS was proposed to solve the research problem, which was to propose an IDS architecture that offers greater robustness and fault-tolerance than the current hierarchical architectures used by nearly all present-day commercial IDSs. The proposed IDS was theoretically described in terms of its main components and how it should function if it were implemented. Thereafter, the proposed IDS was theoretically evaluated in Chapter 6.

The proposed IDS solved the research problem by eliminating single points of failure in the IDS, and also by making the critical components of the IDS non-static. By designing the IDS in such a way that there exists no single component that is essential for the system to detect intrusions, the system can continue to detect intrusions even when most of its components have been attacked or have failed; the IDS is fault-tolerant. By making the critical components of the IDS non-static, the IDS is less vulnerable to direct attack because its components are more difficult to find and consequently are more difficult to disable than in a hierarchical IDS architecture that only makes use of static components.

There are also several unsolved aspects of the IDS, which have been discussed in Section 6.5 together with other significant issues that will need to be considered when implementing the IDS. For example, there is still no optimal solution to ensuring that an attacker will not exploit the function of agent destruction, which is necessary to control the number and types of mobile agents present in the network (see Section 6.5.7).

7.1 Recommendations for further research

Although the proposed IDS provides greater fault-tolerance, the solution also has several drawbacks. The system also has several unsolved aspects, which were beyond the scope of this dissertation. Further research is certainly needed to eliminate or

minimize the effects of the mentioned drawbacks and also to provide satisfactory solutions to the unsolved aspects of the system.

In the proposed IDS, potential intrusions are detected by comparing sequences of system calls generated by an executing privileged program with those sequences that are considered as part of the normal behaviour of that program. This limits the types of intrusions that the IDS can detect (see Section 6.4.1). Therefore, a possible area of future research is to improve the intrusion-detection capability of the proposed IDS. This can be done by increasing the number of intrusion-detection methods supported by the IDS. A variety of new types of mobile agents can be proposed, which use different techniques to detect intrusions.

For example, the proposed IDS cannot detect intruders that use another user's account, because the IDS cannot distinguish between different users. To provide this ability in the IDS, new types of mobile agents could be proposed, which would be able to distinguish between normal and abnormal user behaviour. This would also require that the normal, or expected, behaviour of users be defined by collecting data relating to the behaviour of legitimate users over a period of time. The mobile agents can then apply statistical tests to the observed behaviour to determine the legitimacy of the user behaviour.

A natural progression from this work would be to improve the proposed IDS by providing it with an intrusion-response function. As in the immune system, such an IDS would not only be capable of detecting intrusions, but would also be able to combat them.

Providing an intrusion-response function in the IDS would require a study to determine the types of intrusions that the proposed IDS can detect, as well as a study of the existing techniques or mechanisms that are appropriate for combating each type of intrusion detectable by the IDS. Thereafter, an appropriate mechanism, which is to provide the intrusion-response capability in the IDS, would need to be proposed.

The intrusion-response function could be implemented in mobile agents that would be specialized to respond to specific intrusions. When an intrusion-detection mobile agent

would detect a potential intrusion, it would activate the appropriate intrusion-response mobile agent. The intrusion-response mobile agent would then take the necessary action to respond to the intrusion. By having separate intrusion-detection and intrusion-response mobile agents, the size of individual mobile agents would be small enough to make it practical for them to be transported in the network.

The intrusion-response function could also be implemented in the same agents that detect intrusions. However, this approach would create greater amounts of network traffic because, by combining a specific intrusion-detection function and the related intrusion-response function into a single mobile agent, the size of the mobile agent would be increased. In addition, it is not necessary that the intrusion-response function be transported together with the intrusion-detection function, because the intrusion-response function is only needed when an intrusion has been detected. That is, the intrusion-response function should be transported only when it is required, so as to minimize the amount of network traffic generated by the IDS.

In conclusion, this dissertation has shown that, by studying the human immune system and by applying its properties to intrusion detection, an improved IDS has resulted. This suggests that, even though there are many differences between living organisms and computer systems, the similarities could indeed point the way to improved computer security and computer systems in general.

BIBLIOGRAPHY

Anderson, D., Frivold, T., Valdes, A. 1995. Next Generation Intrusion Detection Expert Systems (NIDES): A Summary. Technical Report SRI-CSL-95-07. Computer Science Laboratory, SRI International.

Bace, R. 2000. *Intrusion Detection*. Macmillan Technical Publishing. Indianapolis, USA.

Bace, R., Mell, P. 2001. Intrusion Detection Systems. Special Publication 800-31, National Institute of Standards and Technology (NIST).

Biermann, E., Cloete, E., Venter, L. 2001. A comparison of intrusion detection systems. *Computers & Security*, 20(8): 676-683.

Botha, M. 2004. NeGPAIM: A model for the proactive detection of information security intrusions, utilizing fuzzy logic and neural network techniques. Thesis, Port Elizabeth Technikon, South Africa.

Bradshaw, J. 1997. An introduction to software agents. In: Bradshaw, J. (Editor), *Software Agents.* AAAI Press / The MIT Press, Chapter 1.

CERT, 1997. CERT Advisory CA-1997-28: IP Denial-of-Service Attacks.

CIDF, 1999. Common Intrusion Detection Framework. Available at: http://www.isi.edu/gost/cidf/ (Accessed in April 2004).

Davies, H. 1997. *Introductory Immunobiology*. Chapman & Hall.

De Castro, L., Von Zuben, F. 1999. Artificial Immune Systems: Part I - Basic Theory and Applications. Technical Report TR - DCA 01/99. Available at: ftp://ftp.dca.fee.unicamp.br/pub/docs/vonzuben/lnunes/trdca0199.pdf (Accessed in July 2003).

Forrest, S., Hofmeyr, S., Somayaji, A., 1997. Computer immunology. *Communications of the ACM*, 40(10): 88-96.

Forrest, S., Hofmeyr, S., Somayaji, A., Longstaff, T. 1996. A sense of self for Unix processes. In *Proceedings of the 1996 IEEE Symposium on Computer Security and Privacy*. pp.120-128.

Foukia, N., Hulaas, J., Harms, J. 2001. Intrusion detection with mobile agents. In *Proceedings of the 11th Annual Internet Society Conference (INET 2001)*, Stockholm, Sweden.

Frank, J. 1994. Artificial intelligence and intrusion detection: current and future directions. In *Proceedings of the 17th National Computer Security Conference*, Baltimore, USA.

Heberlein, L., Dias, G., Levitt, K., Mukherjee, B., Wood, J., Wolber, D. 1990. A network security monitor. In *Proceedings of the 1990 IEEE Symposium on Research in Security and Privacy*. pp. 296-304.

ISS (Internet Security Systems), 1998. Network- vs. host-based intrusion detection. Available at: http://documents.iss.net/whitepapers/nvh_ids.pdf (Accessed in June 2003).

Jansen, W. 1999. Mobile agents and security. In *Proceedings of the 1999 Canadian Information Technology Security Symposium*.

Jansen, W. 2002. Intrusion detection with mobile agents. *Computer Communications*, 25(15): 1392-1401.

Jansen, W., Karygiannis, T. 1999. Mobile agent security. Special Publication 800-19, National Institute of Standards and Technology (NIST).

Jansen, W., Mell, P., Karygiannis, T., Marks, D. 1999. Applying mobile agents to intrusion detection and response. Interim Report 6416, National Institute of Standards and Technology (NIST).

Jansen, W., Mell, P., Karygiannis, T., Marks, D. 2000. Mobile agents in intrusion detection and response. In *Proceedings of the 12th Annual Canadian Information Technology Security Symposium*, Ottawa, Canada.

Kim, J., Bentley, P. 1999. The human immune system and network intrusion detection. In *Proceedings of the 7th European Congress on Intelligent Techniques and Soft Computing (EUFIT '99)*, Aachen, Germany.

Ko, C., Fink, G., Levitt, K. 1994. Automated detection of vulnerabilities in privileged programs by execution monitoring. In *Proceedings of the 10th Annual Computer Security Applications Conference*. pp. 134-144.

Lane, T., Brodley, C. 1997. An application of machine learning to anomaly detection. In *Proceedings of the 20th National Information System Security Conference*. pp. 366-380.

Lange, D., Oshima, M. 1998. *Programming and Deploying Java Mobile Agents with Aglets.* Addison-Wesley.

Lundin E., Jonsson, E. 2002. Survey of Intrusion Detection Research. Technical Report Nr. 02-04. Chalmers University of Technology, Göteborg, Sweden. Available at: http://www.ce.chalmers.se/staff/emilie/papers/Lundin_survey02.pdf (Accessed in July 2003).

Lunt, T., Jagannathan, R. 1988. A prototype real-time intrusion detection expert system. In *Proceedings of the 1988 IEEE Symposium on Security and Privacy*. pp. 59-66.

Mukherjee, B., Heberlein, T., Levitt, K. 1994. Network intrusion detection. *IEEE Network*, 8(3): 26-41.

Ptacek, T., Newsham, T. 1998. Insertion, evasion and denial of service: eluding network intrusion detection. Technical Report, Secure Networks, Inc.

Rothermel, K., Schwehm, M. 1998. *Mobile Agents.* In Kent, A., Williams, J. (Editors) *Encyclopedia for Computer Science and Technology*. M. Dekker Inc. New York, USA.

Sebring, M., Shellhouse, E., Hanna, M., Whitehurst, R. 1988. Expert systems in intrusion detection: a case study. In *Proceedings of the 11th National Computer Security Conference*. pp. 75-81.

Silberschatz, A., Galvin, P. 1999. *Operating System Concepts.* 5th edition. John Wiley & Sons.

Smaha, S. 1988. Haystack: an intrusion detection system. In *Proceedings of the 4th Aerospace Computer Security Applications Conference*, Orlando, USA. pp. 37-44.

Smith, J. 1988. A survey of process migration mechanisms. *Operating Systems Review*, 22(3): 28-40.

Somayaji, A., Forrest, S. 2000. Automated response using system-call delays. In *Proceedings of the 9th USENIX Security Symposium*, Denver, USA. pp. 185-197.

Somayaji, A., Hofmeyr, S., Forrest, S. 1997. Principles of a computer immune system. In *Proceedings of Meeting on New Security Paradigms*, Langdale, UK. pp. 75-82.

Spafford, E., Zamboni, D. 2000. Intrusion detection using autonomous agents. *Computer Networks*, 34(4): 547-570.

Sundaram, A. 1996. An introduction to intrusion detection. *Crossroads: The ACM Student Magazine*, 2(4).

Tan, K., Maxion, R. 2002. Why 6? Defining the operational limits of stide, an anomaly-based intrusion detector. In *Proceedings of the 2002 IEEE Symposium on Security and Privacy*, Berkeley, USA. pp 188-201.

Warrender, C., Forrest, S., Pearlmutter, B. 1999. Detecting intrusions using system calls: alternative data models. In *Proceedings of the 1999 IEEE Symposium on Security and Privacy*. pp. 133-145.

White, G., Fisch, E., Pooch, U. 1996. Cooperating security managers: a peer-based intrusion detection system. *IEEE Network*, 10(1): 20-23.

Zielinski, M., Venter, L. 2004. Applying similarities between immune systems and mobile agent systems in intrusion detection. In *Proceedings of the 2004 Information Security South Africa (ISSA) Conference*, Midrand, South Africa.