2001

Mobile agent-based attack-resistant architecture for Distributed Intrusion Detection system

Sentil Kumar Selliah
West Virginia University

Mobile Agent Based Attack Resistant Architecture for Distributed Intrusion Detection System

Sentil Selliah

Thesis Submitted to
College of Engineering and Mineral Resources
at West Virginia University
In Partial Fulfillment of the Requirements
For the Degree of

Master of Science
In
Computer Science

Approved By

Bojan Cukic, Ph.D., Committee Chair

V. Jagannathan, Ph.D.

Todd Montgomery

Lane Department of Computer Science and Electrical Engineering
Morgantown, WV

2001

Keywords : Computer Security, Network Security, Intrusion Detection, Attack Resistant Architecture, Mobile Agents

ABSTRACT

Mobile Agent Based Attack Resistant Architecture for Distributed Intrusion Detection System

Sentil Selliah

The majority of the Distributed Intrusion Detection systems lack measures for providing security and integrity to their own components. The hierarchical organization and the static nature of the intrusion detection components in a largely distributed environment make them the likely targets of attacks. By disabling few operationally critical components along the hierarchy, an attacker can succeed in disabling the system's capability to correctly detect intrusions. One solution to this problem is to eliminate the system components' static nature by wrapping them as mobile agents. Through mobility we achieve an attack resistant architecture for the hierarchical distributed intrusion detection components. As mobile agents, these components can hide in a complex network topology, constantly roaming to avoid detection, and be replaced when compromised. In this thesis we analyze an approach where mobile agents replace the static internal components of a hierarchical distributed intrusion detection system.

We developed a system for this model using IBM's Java based mobile agent (Aglet) framework with the following features: randomized agent locations, decoy agents to allude an attacker from functionally critical components, a redundant polling mechanism to ensure the integrity of mobile agents' data processing and a mechanism for the mobile agents' to avoid malicious hosts.

Acknowledgements

First of all I would like to thank Dr. Bojan Cukic, committee chair, for his support, advise and for providing me the opportunity to work with him at West Virginia University.

A special thank you for Todd Montgomery for providing me with valuable comments throughout my research. He's been a source of great motivation during my college life at West Virginia University. I would also like to thank Dr. Jagannathan for his support and providing me with valuable knowledge.

Finally I would like to thank my special friends Annie, Shawn, Karthik and Vinay, who provided me nothing but encouragement and support during busy and stressful moments of thesis research.

Dedication

In dedication to my

Mom

Dad

Brother, Ashok

Sister, Suda

For this sweet moment

Table of Contents

ABSTRACT ii

ACKNOWLEDGEMENTS iii

DEDICATION iv

TABLE OF CONTENTS v

LIST OF FIGURES vii

1. INTRODUCTION 1

1.1 COMPUTER SECURITY 1
1.2 INTRUSION DETECTION SYSTEM 3
1.3 DISTRIBUTED INTRUSION DETECTION SYSTEM (DIDS) 3
1.4 SECURITY OF DISTRIBUTED IDS COMPONENTS 4
1.5 THESIS STATEMENT 6

2. BACKGROUND AND RELATED WORK 7

2.1 INTRODUCTION 7
2.2 INTRUSION DETECTION 7
2.3 MOBILE AGENTS 9
2.4 RELATED SYSTEMS 11
2.4.1 Bro 11
2.4.2 The Mobile Agent Attack Resistant Distributed Hierarchical Intrusion Detection System 13
2.4.3 RealSecure 14
2.4.4 An Architecture for Intrusion Detection using Autonomous Agents (AAFID) 15

3. THE ATTACK RESISTANT ARCHITECTURE 17

3.1 INTRODUCTION 17
3.2 COMMUNICATION (AUTHENTICATION, ENCRYPTION AND RELIABILITY) 17
3.3 LOCATION TRANSPARENCY 19
3.4 DESTRUCTION OF COMPONENTS 20

4. MOBILE AGENT BASED ATTACK RESISTANT ARCHITECTURE FOR A DISTRIBUTED INTRUSIONDETECTION SYSTEM 23

4.1 INTUITION BEHIND OUR APPROACH 23
4.2 MOBILITY OF INTERNAL COMPONENTS 24
4.3 ASSUMPTIONS 25
4.3.1 Redundant Mobile agent (MA) Platforms 25
4.3.2 Secure Public/Private Key Infrastructure 26
4.3.3 Compromising MA Platforms 26
4.3.4 Malicious Leaf Node Components 26
4.4 SYSTEM ARCHITECTURE 27
4.4.1 Randomized Agent Location 27
4.4.2 Agent Communication 29
4.4.3 Decoy Agents 30
4.4.4 Mobile Agent Security 31
4.4.5 Avoiding Malicious Host 34

5. SYSTEM IMPLEMENTATION 35

5.1 INTRUSION DETECTION TECHNIQUE 35

5.2 MOBILE AGENT PLATFORM 37
5.3 NETWORK INTRUSION DETECTION SYSTEM AS LEAF NODES 37
5.4 SCANNING TOOL 38
5.5 TEST BED 38
5.6 DISCUSSION 44
6. CONCLUSION AND FUTURE WORK 48
REFERENCES: 50
APPENDIX A 53
APPENDIX B 57
APPENDIX C 59

List of Figures

Fig 1. The Architecture of a Hierarchical Distributed Intrusion Detection System..... 5
Fig 2. Physical Layout of the System. 25
Fig 3. Randomization of Agent Locations with in a single network 28
Fig 4. Agent Communication..... 30
Fig 5a. Multiple agents submitting their results for Voting Server. 32
Fig 5b. Voting on the Voting Server..... 32
Fig 5c. Voting Server updating Agent/Host Trust Level. 33
Fig 6. GrIDS tree for Scan Attack..... 36
Fig 7. System test bed 39
Fig 8. Attack Tree of MA platform 1..... 42
Fig 9. Attack Tree of MA platform 2..... 43
Fig 10. Root node's generation of an attack tree 44
Fig 11. Aglet Framework..... 54

1. Introduction

1.1 Computer Security

Web site defacements, Worm attacks and Denial of Service attacks have become headline topics in many technical news sources recently. Over the passed couple of decades attacks on computer infrastructures have increased sharply, in pace with the booming growth of the Internet. The increased deployment of computer networks over the Internet by commercial organizations, academic institutions and individuals have provided a playing field for intruders carrying out attacks on computer systems since many of the deployed systems are vulnerable to attacks. Increased sophistication of attack techniques and the widespread availability of automated intrusion tools pose a challenging threat to the security of computer network infrastructures. As a result Computer Security, largely neglected a decade ago, has become the focal point of many commercial as well as academic institutions. Computers lacking security on the Internet are targets for widespread attacks as well provide a gateway to carry out attacks on other computer systems.

Generally speaking secure computer system is defined "as one that can be depended upon to behave as it is expected to" [GS96]. The expected behavior of a secure computer system would be in accordance with its specified security policy [San95]. Intrusion into a computer system is seen as a violation of the security policy in place. The above definition can be refined to reflect three important attributes of a secure computer behavior. They are confidentiality, integrity and availability.

Confidentiality ensures that the system information is only accessible to authorized persons. Integrity ensures that the information is not altered by unauthorized persons and not detectable by the authorized one. Availability ensures the accessibility of system resources for authorized users whenever necessary.

Threats to a computer system can come in different forms. Simple scripts that exploit system vulnerabilities to elevate privileges, unsupervised virus and worm programs capable of performing undesired damages to the system and multiplying to affect other systems, and very sophisticated distributed/coordinated attacks that render a system unavailable are few forms. A secure system should posses the ability to prevent any attacks aimed at themselves. If the system fails in the prevention mechanism it should be capable of detecting any forms of security violation.

At present there are several mechanisms in place to ensure the security of a computer system. Many systems utilize an authentication model as a first line of defense. This model provides access to a system's resources only to a trusted domain. But it does not restrict a trusted member of the domain on activities that could be performed on the resources. An intruder could penetrate a system by circumventing these authentication and access control mechanisms by exploiting bugs in the operating system software and application software that run on the computer system. Unless perfect bug free software is developed it would be impossible to prevent any intrusions into a system. Hence computer systems cannot solely depend on access control and flow control mechanisms to provide a blanket of security. Hence there should exist other measures that are capable of identifying security breaches or recognizing any measures that are indicative of a pre-

attack scenario and reporting these events to the system security officer. Intrusion detection server's this purpose by forming the last line of defense against the intruders.

1.2 Intrusion Detection System

An *Intrusion Detection System* (IDS) is responsible for performing Intrusion Detection within a system. A computer requires the analysis of it's data and activities individually to detect any intrusions into the system. However the detection of coordinated and distributed attacks requires the collective analysis of information gathered at various locations off of a number of systems. Because the nature of the coordinated and distributed attacks is that its ability to be a normal behavior on an individual nature and of an anomalous behavior on a collective nature. Hence, a system (*Distributed Intrusion Detection System*) responsible for detecting distributed attacks requires components that spread across different (networks of) computers which build the distributed environment.

1.3 Distributed Intrusion Detection System (DIDS)

The first generation of *Distributed Intrusion Detection Systems (DIDS)* followed a simple, so called *two component model,* with sensor components collecting the data at various points and a centralized component performing analysis and detection. This model, although suitable for small systems, does not scale well for large distributed systems, since the centralized analysis component has to deal with large volumes of data processing. Hence, the later versions of DIDS introduced middle layers that are responsible for data reduction and analysis (see Figure 1). A tree like hierarchical

organization is preferred for these components. Consequently, functional components of a DIDS can be categorized into three layers:

- Leaf Nodes (Sensors)

  Leaf Nodes are responsible for collecting data in their local computer/network. Input to these nodes is any information that could lead to the detection of intrusions. For example log files, network packets and system call traces are some of the examples. The data collected by leaf nodes is sent to the internal nodes.

- Internal Nodes

  These nodes are responsible for analyzing and reducing the volume of data received from one or more sensors. The analyzers pass along the further reduced information relevant for intrusion detection to the root node.

- Root Node (Command and Control Point)

  The root node is a centralized processing unit that is responsible for identifying any intrusions based on the data it receives from the analyzers. The root node carries out active and/or passive counter measures in cases of positive detection.

1.4 Security of Distributed IDS components

We expect a system that forms a last line of defense against attackers to be secure itself. But the security of DIDS components has become an exception rather than the rule. The static nature of the components of an intrusion detection system are vulnerable to attacks themselves.

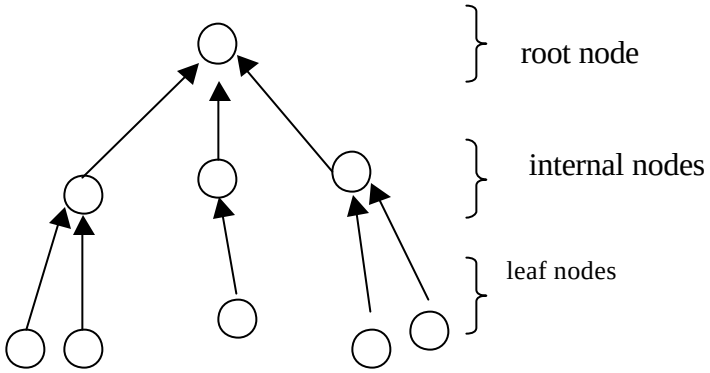

Fig 1 The Architecture of a Hierarchical Distributed Intrusion Detection System

In fact, components of intrusion detection systems are popular targets for attackers. Especially in a distributed environment, if an attacker disables few components along the hierarchy of an intrusion detection system, the result is the reduced detection capability or, in the worst case, the elimination of intrusion detection. As a result, an intruder could invisibly penetrate the system by circumventing the intrusion detection capability. This inherent weakness of most commercially available intrusion detection systems is a result of their hierarchical and static organization. At the same time, organizing components into a hierarchical architecture provides many performance and organizational advantages, including efficiency and flexibility. The alternative approach of organizing intrusion detection components into a non-hierarchical architecture, where the components of the DIDS are completely distributed in nature, has proven inefficient both in detecting and reporting attacks [PM99]. Majority of the current research in intrusion detection is focused on the intrusion detection techniques, including data mining and intrusion reporting. Protecting components of an intrusion detection system from attacks

or identifying and reconstructing the components that have been disabled to ensure the proper functionality of the intrusion detection has received limited attention [Axe99, Hak99].

1.5 Thesis statement

In this thesis we attempt to provide an attack resistant architecture for intrusion detection systems, while keeping intact the hierarchical organization. An attack resistant nature provides the DIDS components the much needed security to exist in a distributed environment. At the same time providing ability for the root node to identify and destroy a malicious component without affecting the critical functionality of the overall system. We achieve this goal by eliminating the static nature of the internal components and wrapping them as *mobile agents* (MA). As MA's the IDS components can evade attacks. The constant movement of these agents in a network among multiple hosts makes it difficult for an attacker to locate and disable them. Further, these agents could obtain information about malicious hosts and avoid them in the network. So called decoy agents, explained later, can insert junk traffic into the network to distract the attacker off of the critical functional components.

The rest of the thesis is organized as follows. Chapter 2 discusses related work. Chapter 3 outlines the architectural features of the proposed attack resistant architecture for distributed intrusion detection systems. Chapter 4 describes some of the critical issues underlying the proposed design, such as agent communication, location, etc. Chapter 5 outlines appropriate methodology for system implementation. We conclude with a summary and an overview of further work in Chapter 6.

2. Background and Related Work

2.1 Introduction

This chapter provides background information on concepts related to our thesis and describes several Intrusion Detection systems that provide solutions to the prevention of attacks on the intrusions detection system components. We call the latter attack resistance capability of an ID system. Many research and commercial entities have released prototypes or complete versions of distributed intrusion detection systems whose hierarchical organization poses a major threat to the system's integrity. Lack of emphasis on the ID system's security and various operating system functional discrepancies have led to attacks that damaged, evaded and misdirected intrusion detection systems. A recent survey [Axe99] selects only two out of twenty systems with moderate or high security features. A recent survey of commercial ID systems [Hak99] provides similar results.

2.2 Intrusion Detection

An *intrusion* is defined by Anderson [And80] as the deliberate unauthorized attempt to :

- access information,
- manipulate information, or
- render a system unreliable or unusable.

An intrusion violates the security policy of a system in place. *Intrusion detection* is the process of detecting intrusive activities through the observation of an attack in progress or from recognizing the effects of an intrusion after the fact. The techniques used to detect intrusions on a system require the knowledge of either the expected behavior of the system or the attacks that could be carried out against the system. Intrusion detection techniques can be categorized into two main groups: *anomaly intrusion detection* and *misuse intrusion detection* [MHL94]. Anomaly detection techniques identify any unacceptable deviation from expected behavior of an individual user or an application. The expected behavior is defined in advance, in manually or automatically developed profiles. This is then compared with the current activities of the user or the application. An uncharacteristic deviation would be an indication of an intrusion. For example, if a user typically uses his/her account to check mail and run a Web browser, an activity on his/her account that runs large programs is anomalous and, hence, might be the result of an intrusion. Anomaly detection attempts to compare the current and the expected user behavior and flags unexpected/atypical behaviors as potentially intrusive. [AL95] describes a statistical approach used for Anomaly Detection.

On the other hand misuse intrusion detection refers to the analysis of certain well-defined patterns of attacks that exploit weaknesses in system and application software. For example, packets of network traffic could be analyzed for a series of characters, which could represent a signature of an attack sequence. This mechanism requires the knowledge of unacceptable behavior to detect an intrusion as opposed to anomaly detection, which is based on the identification of normal behavior.

An *Intrusion Detection System* uses either one of the above techniques or both of them together to detect any intrusions into the computer system.

2.3 Mobile Agents

Mobility of the DIDS components are important part of our solution. This is achieved through wrapping these components as mobile agents. One possible definition for a *Mobile Agent* is :

'A computational entity, which acts on behalf of others, is autonomous, proactive and reactive and exhibits capabilities to learn, cooperate and move'. Since roots of this definition come from distributed intelligent systems, many argue that a basic agent model is not required to exhibit any learning capability. As a result there exist many definitions for a *mobile agent*. However based on the experience with respect to our project, we adopt the above definition minus the capability to learn (not intelligent).

A framework is required for a system to incorporate Mobile Agents (Refer to Appendix A for a description on Java based Aglet Framework Model). In addition to providing the basic agent model, the framework is required to provide the following:

1. Life-cycle Model: Create, start, dispose, stop, suspend and destroy an agent.
2. Computational Model: ability for an agent to perform data processing and manipulation.
3. Security Model: Agents' ability to access system resources and system's ability to access agent's results securely.
4. Communication Model: Ability to communicate with other agents.

The general advantages of using *mobile agents* are numerous including reduced network latency, reduction of network load and autonomous execution. Hence, there are many applications that could utilize mobile agents. However ensuring the security and reliability of the agents that circulate the network and the security of the remote hosts (host providing agent framework) has become very challenging and a major restraining factor for the deployment of agent based technologies. There are two different categories of threats: threats stemming from an agent attacking an agent platform and an agent platform attacking agents [JK99].

Agent attacking the Platform

In this category of attacks an agent could exploit any security weakness that may exist in the host platform. Denial of service attacks, masquerading attacks and accessing unauthorized information are few threats a platform could face by malicious agents. In denial of service attacks the agents could consume excessive amount of platform resources and render the platform unavailable to other agents. In a masquerading attack the agent could imitate another agent and gain privileges to access the platform's resources which is not entitled to. These attacks might cause information loss, operational failures and even modification of system critical information.

Platform attacking agents

The mobile agents hop from one host to the other with the static code, accumulated states and data. Since the host provides the execution environment for the agents, the host has complete control over it, and can use it to analyze or modify the code and the results or, worse, destroy the agents.

Denial of service attacks, alteration of agent code, masquerading attacks are some of the threats posed by a host to the agents. In the denial of service attacks a malicious host could deny resources to the agent that are necessary for it's execution. The host could masquerade as a trusted host to attract agents and then analyze or modify their code.

Various studies have discussed solutions to protect a host from attacks by agents. Some of the proposed solutions include execution of digitally signed agents, isolated execution of agent code, proof carrying code, authorization and attribute certificates. Research and solution for protecting agents from malicious host is in it's infancy stage due to it's complex nature. Since it's very hard to protect an entity which runs under another entity's authority. There are even arguments that a general solution for these kind of threats are impossible. However there exists very complex application specific solutions like partial result encapsulation, computing with encrypted function, obfuscated code and execution tracing.

2.4 Related Systems

2.4.1 Bro

The Bro [Ver88] is a standalone system capable of detecting intrusions by passively monitoring network activities. The Bro system can be divided into three distinct layers. Each layer would responsible in performing data reduction. The lower layer captures raw network packets for further processing. These packets are passed onto the middle layer, which is the so-called event engine. The event engine decides whether the packet should be logged or not based on certain control connection state handlers. The top layer is the scripting interface that enables the addition of new control handlers. The

paper presenting the Bro system addresses the problems of the attacks the monitor could withstand.

Three categories of attacks are mentioned by the author that could be carried out against the Bro system and mechanisms that are adopted to detect and prevent these attacks[Ver88].

Overload Attacks

The goal of this attack is to overburden the monitor to a point where it fails to keep up with the data stream it needs to process. Hence failing to detect an intrusion afterwards. The Bro system handles these types of attacks by shedding some load when it becomes heavily stressed. logging at regular intervals, how many packets it has dropped and processed, which in turn could be used to determine the anomalous nature of the network traffic.

Crash Attacks

The purpose of these attacks is to stop the monitor altogether. These attacks could be carried out by analyzing the source code for any vulnerability due to coding errors or by exhausting the monitor's resources completely. Careful coding practices and extensive testing would elevate the first type of attacks. In order to defend against the second type of crash attack the Bro utilizes a watch dog monitor which expires every T seconds. Upon expiration the monitor checks to see whether the system has failed to process any packets in the previous T seconds slot. If so the monitor assumes a processing jam and stops the system for post mortem analysis. During this time the monitor logs the packets. Hence any intrusion that follows after a crash attack is recorded.

Subterfuge Attacks

These attacks involve misleading the monitor as to the meaning of traffic it analyzes. These attacks are the hardest to detect. The author claims that certain assumptions that were taken at the time of development in order to prevent these types of attacks. The author lists several of these attacks but no claims have been made on whether these attacks were tested against Bro and been dealt with.

2.4.2 The Mobile Agent Attack Resistant Distributed Hierarchical Intrusion Detection System [*]

Mell and McLarnon [PM99] proposed a methodology for providing attack resistant capabilities for hierarchical ID systems. This is achieved by wrapping the non-leaf components of an IDS as mobile agents (MA), hence eliminating the static nature of these components and providing an attack resistant capabilities to the IDS. The ability of the mobile agents to move between different platforms and avoid attacks is the focal point of the system. The authors divide the techniques used to implement attack resistance into five stages.

Randomizing agent locations

The mobile agents continuously move between networks by randomly selecting an agent platform among several that are available. The continuous and random movement of these agents around the network will complicate attacker's intention of pinpointing agent location to destroy it. The authors mention that an avenue of attacks against system would be to monitor agent communication between agent platforms and analyze the underlying transport header information to determine the number of agents that exist in certain platforms and the agent platforms they desire to communicate with.

[*] We incorporated few ideas into the development of our system from this proposal.

However, the authors claim that the attacker would require sufficient sniffing points to do so.

Removing Centralized Directory Services

The authors claim that an attacker could find and destroy the centralized directory services that are necessary for the proper functioning of the agents. Eliminating these single points of failure would be of complicated task. The authors propose a concurrent negotiation algorithm for inter agent communication that would eliminate a centralized directory service. This algorithm requires an agent to buddy up with another agent at the time of its deployment. Then they constantly notify each other about their where about.

Evading Attackers

Destroying an agent platform may lead to the death of an agent. In this case the authors mention that the buddies of the agents should detect this event and avoid network locations suspected to host attackers and inform other agents about the event.

Resurrecting Killed agents

In case of an agent being killed they are substituted by back up agents to hold their positions. These back up agents are constantly updated with state information of the active agent. In case of an active agent's death the backup agents negotiate among themselves and pick a candidate for replacement.

2.4.3 RealSecure

RealSecure [ISS01] is a commercial distributed intrusion detection system developed by Internet Security Systems (ISS). The documentation does not provide sufficient information about securing components of the intrusion detection system. However, a survey done by [Hak99] sheds some light onto the defensive measures taken

by RealSecure. The RealSecure consists of RealSecure Engines, RealSecure Agents, and RealSecure Manager. The system uses encrypted communication channels for communication between the distributed agents and the centralized manager. The agents transmit an "I'm alive" heart beat message to the manager at regular intervals to indicate their status. In cases when these messages are not received the manager assumes that the agent is attacked. Obviously, this approach does not consider the possibility of agents being compromised.

2.4.4 An Architecture for Intrusion Detection using Autonomous Agents (AAFID)

Balasubramaniyam et.al. propose an architecture [BG98] for building Intrusion Detection Systems. that use agents as their lowest level element for data collection and analysis and employs a hierarchical structure for scalability. A simple example of a system that adheres to the AAFID architecture consists of the following three essential components: agents, transceivers and monitors. Here agents are not necessarily mobile but certainly autonomous. An AAFID system can be distributed over any number of hosts in a network. Each host can host many agents that monitor for interesting activities in that host. Agents report their findings to a single transceiver. Transceivers are per-host entities that monitor the operations of agents and control their functionality running in their host. Finally the transceivers report their results to one or more monitors. Each monitor oversee several transceivers. They perform higher level correlation on the results obtained from the transceivers, to detect any distributed attacks. The authors have raised some issues regarding the reliability and security of the architecture’s inter-host communication (Communication between different components of the architecture).

Reliability

Reliability determines that the messages from one host to another arrive correctly and on time. The authors point out that the reliability factor may or may not produce drastic impact on the operation of an IDS in terms of the ability to detect intrusions. Further they claim the meaning of “acceptable” loss of messages depends on the deployed system.

Security

The authors claim that privacy and authentication are two important needs for an IDS, Since they would generate sensitive information about hosts. Attacker should not posses the capability to generate false messages that are accepted as legitimate. Cryptography is seen as the proper solution to prevent these malicious activities with the added cost in performance and overhead imposed on the system. Further the authors mention the denial of service attacks through which an attacker could prevent the ID system from accepting legitimate messages.

The authors stop short of providing any definite conclusions instead they discuss about three questions. They are ‘is privacy necessary?’, ‘is authentication necessary?’ and ‘how to implement them?’.

This system mainly deals with the security of the communication mechanism on the IDS components, but lacks discussions and suggestions on the component’s security itself.

3. The Attack Resistant Architecture

3.1 Introduction

The components of a Distributed Intrusion Detection System (DIDS)[1] are spread across network topology that builds a distributed environment. Their ability to properly function is vital for the detection of coordinated attacks that are carried out against the distributed system. Failure of some of these components would reduce or, in the worst case, eliminate the detection capability of the DIDS. As a result, an attacker could penetrate the system and avoid being detected. Hence, functionally critical DIDS components are favorite targets for attacks. In this section we elaborate measures that could be taken by attackers to disable the DIDS components and prevention mechanisms to survive these attacks.

3.2 Communication (Authentication, Encryption and Reliability)

The hierarchical nature of the DIDS largely depends on the communication between components for information accumulation that is required to detect intrusions and to implement countermeasures. Insecure and unreliable communication will hinder the accurate detection of intrusions. The data flows upward the hierarchy from lower nodes to upper level nodes. Control information flows from the root node to the lower nodes. An attacker may explore avenues for disrupting the communication mechanisms and even attempt to analyze traffic flow between DIDS components.

[1] We only refer to DIDS whose components are hierarchically organized. Refer Chapter 1 for a description of hierarchical organization of DIDS components.

Analyzing the contents of communication traffic might reveal sensitive information about host that are being monitored as well as the DIDS components' logistic information. Hence it becomes necessary that the DIDS components adopt a secure communication mechanism. We restrict our discussion only to the underlying network transport security. Physical security of the communication links is beyond the scope of this document. This disable an attacker during his reconnaissance phase. Encryption and authentication of messages that are interchanged between the DIDS components would provide a degree of privacy and disable attempts by an attacker to generate false messages that are accepted by the components as legitimate.

The strength and success of encryption and authentication depends on the underlying algorithms and the distribution of keys involved. Although encryption and authentication sound promising in providing secure component communication, they impose a significant cost and overhead on the systems. Since encryption and authentication involves, performing complex mathematical operations. Hence, a careful analysis of requirements for secure communication is deemed necessary since reduced performance of a system affects the timely processing and submission of intrusion warnings.

Loss of messages may be a significant factor in the functioning of a DIDS. A DIDS monitoring a time critical high assurance system might not tolerate even a small amount of message loss. But on the other hand a DIDS monitoring distributed web servers could accommodate a certain level of message loss and still perform detections at an acceptable rate. Hence requirements for reliable communication will vary from system to system.

Providing secure and reliable solutions for communication can be part of the component itself or can be provided by the network transport protocol that is being operated on. Widely used transport protocol, Transmission Control Protocol (TCP) [Pos81], provides reliability for the data being transferred but lacks the encryption or authentication capability. TCP combined with secure Internet Protocol (IP-SEC) [AK98] can provide the additional encryption and authentication capabilities for communication. Alternatively, the use of these well established data transport mechanisms could be substituted with a customized transport protocol for DIDS component communication. But it comes with the complexity of developing a new transport protocol.

The above mentioned secure communication mechanisms complicates traffic analysis attacks and masquerading attacks. Denial of service attack is another type of attacks that can be carried out against the components. This attack is carried out by overloading components into servicing large number of garbage requests. To prevent these kind of attacks very careful measures should be taken to handle unexpected conditions. This could be achieved by careful design and comprehensive testing of the system.

3.3 Location Transparency

Identification of the DIDS components would be the first information an attacker attempts to gather. In a TCP/IP based network, IP address identifies network component's location. Static allocation of DIDS components increases the chances of being detected. In addition, static allocation of DIDS components to provide a convenient scenario to perform reconnaissance operations over a period of time. But the failure to

disclose a component's location to other components of the system would render it unreachable. This defeats the purpose of collaboration of the DIDS components.

Constant movement of the DIDS components solves the location transparency problem. Reasonable frequency of movement may confuse an attacker. But moving components requires that they save their execution state before movement to avoid loss of previous results. It would be inefficient for a large server to constantly move among a set of networks because of it's comprehensive state related information.

Another suitable mechanism is to use multicast for communication. Nature of multicast enables members of a group to join and listen on a multicast group address for messages communicated to that group. Multicast protocols do not require that the sender is aware of individual group members. In contrast point-to-point communication requires that the sender is aware of it's receivers addresses. Multicast is based on the unreliable User Datagram Protocol (UDP) [PS80]. However there exist multicast protocols that support reliable transmissions [MR98]. In order for components to communicate, the message gets send out to the whole group consisting many members, rather than sending to a single member. This would result in unnecessary traffic consuming the network bandwidth. Hence the system should analyze the issues carefully before utilizing the use of movement of components and multicast as a form of communication.

3.4 Destruction of Components

It is a very difficult task to protect DIDS components from attacks since they are spread across a wide area of heterogeneous networks. A best effort scheme should be

adopted to protect the components against intruders. At the same time additional mechanisms should exist to recover the components that are destroyed or warn other components about imminent risks. This to be successfully operational the root security controller should continuously keep track of the 'alive/dead' state of the components. The components of a DIDS are individual programs running on a general purpose computer or a specially dedicated server only performing DIDS activities. A data analysis component may be of a the first form and a root node responsible for detecting intrusions take the latter form. Hence an attack may disable the entire server or the single process or in the worst case control their activities. The functionally critical components like the root node can be strictly protected against attacks by operating it in a secure system. In this section we focus on small system components since they lack very strong security measures.

An attacker killing a component entirely would result in the lack of result submissions from that component. Another technique is to require a component to send out a heart beat message at a regular interval to the root security controller. When this message stops the security controller could assume the destruction of the component. Similar technique is used in RealSecure [ISS01] mentioned in Section 2 of this document. A clever attacker might attempt to take control of the component by intruding the machine it runs on and manipulate or modify it to carry out attacks on other components. Detection and prevention of these kind of malicious activities can be very challenging. Solutions vary from implementing a simple host based intrusion detection system on the machines that host the components to profiling component activities and identifying anomalous behavior. The process of identifying the components' destruction or

modification should be followed by elimination and replacement or repair of the components.

4. Mobile Agent Based Attack Resistant Architecture for a Distributed IntrusionDetection System

In this section we describe our approach to building an attack resistant architecture for a DIDS using mobile Agents.

4.1 Intuition Behind our Approach

As described in chapter 1, a hierarchical ID system consists of three main layers of operational components:

1. Root node functions as a centralized decision-maker.
2. Internal nodes perform data analysis and reduction.
3. Leaf nodes perform data collection.

The root node is a centralized server, which accumulates all necessary information from the internal nodes to make a collective decision on whether an intrusion has occurred or not. In addition, it devises counter measures in the event of an intrusion. Since there is only one root node, taking into account its role, we assume that it can be protected reasonably well to prevent major security infringements by some of the existing methods (trusted server on a trusted network, for example). Our major concern is the security of leaf nodes and the internal components since they are distributed among various (possibly insecure) networks making them vulnerable to attacks. We claim that by sufficiently securing these nodes we can provide an attack resistant IDS architecture operating in a largely distributed environment. In this thesis, we concentrate on the methodology that provides an added level of assurance and survivability for internal components of a hierarchical IDS. Our approach concentrates on the mobility of the internal nodes. At

this time, no provisions are made for the mobility of leaf nodes, since their function is to capture data from certain points on the network.

4.2 Mobility of Internal Components

The concept explored in this document is building the internal components of the IDS utilizing mobile agents (MA) technology. For this purpose, we restrict the definition of mobile agents as follows:

> 'Mobile Agents are autonomous software entities that are capable of moving among platforms in different networks.'

The mobility enables the components to evade attacks by changing their physical locations frequently. The agents also have the capability to avoid network locations that might have been maliciously tampered with, until measures have been taken to evaluate the "attack".

Attack resistant mobile agents operate in an architecture where leaf nodes (referred to from this point on as *monitors*) are network intrusion detection systems (NIDS). These monitors passively observe the network traffic and report any intrusive behavior to the agents, i.e. the middle layer components. In general, different group of agents would be responsible for processing data received from different monitors. The agents try to build a cumulative knowledge of the activities in the networks that come under their supervision. Analyzed data is sent up to the root node, which is responsible for determining whether an intrusion has occurred. Figure 2 illustrates an example of the system architecture.

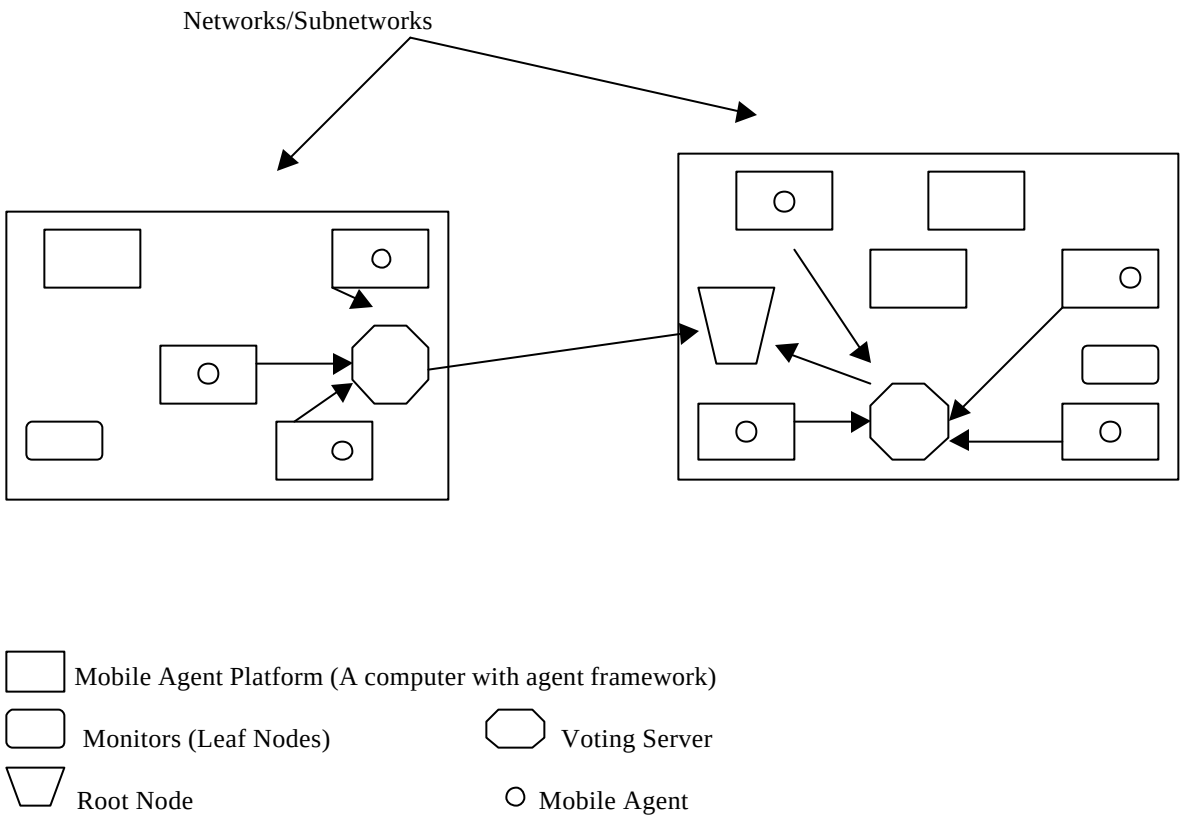

Fig 2. Physical Layout of the System.

4.3 Assumptions

We have adopted several assumptions in the development of our model. It is necessary that the reader perceives our system model in the context of these assumptions.

4.3.1 Redundant Mobile agent (MA) Platforms

The distributed environment where the DIDS is employed consists of redundant MA platforms that are distributed across several heterogeneous networks. These platforms are equally capable of hosting mobile agents and providing all resources necessary for the proper execution. However we do not necessitate MA platforms to

provide a secure execution environment. Refer Section 4.4.4 for more information on agent security.

4.3.2 Secure Public/Private Key Infrastructure

For the purpose of encryption and authentication of messages that are exchanged between components of the DIDS, there exists a secure architecture capable of distributing public and private keys to the DIDS components. This ensures that an attacker is not capable of carrying out attacks against the components by compromising the keys.

4.3.3 Compromising MA Platforms

MA platforms can be a process in a general purpose computer. Hence compromising the system that hosts the MA platform is a possibility. After a system compromise an attacker may or may not have sufficient privileges to shutdown the MA platform on the host. But we assume that the attacker lacks sufficient resources to scan a group of networks to identify all the systems that host a MA platform.

4.3.4 Malicious Leaf Node Components

It is necessary to understand that attacks on the internal components of the DIDS could be carried out directly or through the leaf nodes of the system. An attacker can easily carry out denial of service attacks by overloading the internal components of the system through leaf nodes. Security of the leaf nodes is not discussed in the thesis. We

assume that the leaf nodes are not compromised to carry out attacks against the non-leaf components.

4.4 System Architecture

In this section we provide details regarding the concepts utilized in creating an attack resistant mobile agents based distributed intrusion detection scheme and evaluate it's advantages and drawbacks.

4.4.1 Randomized Agent Location

Mobility of DIDS components[2] immediately eliminates some of the threats posed by attackers. Constant movement enables the agents to indirectly hide their locations and evade being detected. However a predefined pattern of movement would not be much of an attack prevention mechanism since a determined attacker could predict agent locations after some extensive analysis of traffic flow among different platforms. This could be eliminated by random movement of the components.

When the agents are created they are located randomly among different networks and platforms within the distributed system. When they are active in a MA platform they perform analysis and reduction on data received from the monitors. After processing they move to their new location. In the absence of any data transmission from the monitors the agents stay stable in the MA platform until a timer expires.

The choice of the next destination is selected from a list of other MA platforms that is maintained by individual MA platform. The agents can then randomly choose a

[2] In the rest of the section components refer to internal components only, unless stated otherwise.

location of their next destination. The constant randomized movement of the agents is expected to confuse an attacker.

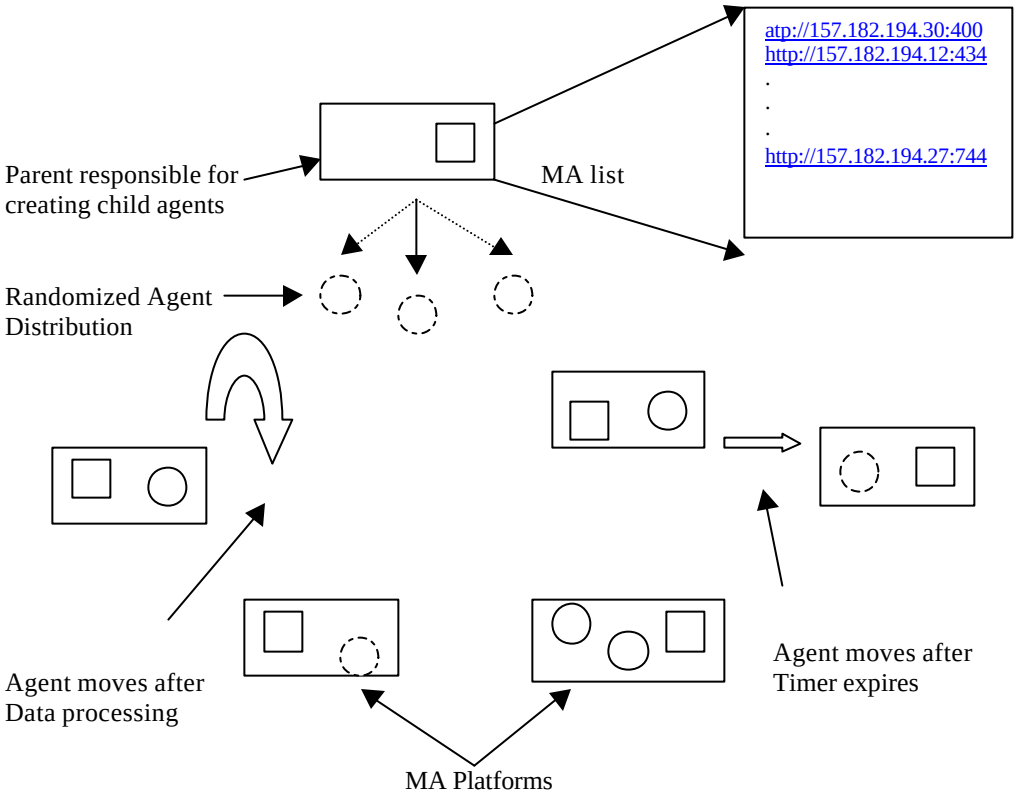

Fig 3. Randomization of Agent Locations with in a single network

One possible avenue of attack against our system could be by obtaining the list of locations/networks from the MA platforms. This information would enable the attacker to target MA platforms and, potentially, hinder the functionality of the agents. We can put in place several measures to identify MA platform compromises. A host based ID system could be installed to monitor any intrusive behaviors in the MA platforms locally. Another protective measure could be populating the MA platform list with many bogus MA platforms that could act as “honey pots”. These “honey pots” could raise their alarm when anything other than a digitally signed mobile agent tries to communicate to it.

If an attacker gains access to the MA platform by surpassing all the measures in place he/she would have the ability to control the activities of the agents and modify the agent code. In the later sections we illustrate a redundant polling mechanism that could mask any incorrect results provided by malicious agents.

4.4.2 Agent Communication

The mobile agents should have a secure communication mechanisms to receive information from the monitors as well as to exchange their results with other components in the hierarchy. The leaf nodes communicate with the agents by multicasting their results to a group address. Hence the leaf nodes do not require any knowledge of the individual agent's locations. The agents responsible for processing this information listen on the same group address as the monitors. This mechanism eliminates security concerns that may arise due to the fact that an attacker's analysis of header information (which is not encrypted on a standard implementation of IPv4) may reveal the agent location. Passive role of the agent (consumption of monitor's report only), makes them "invisible".

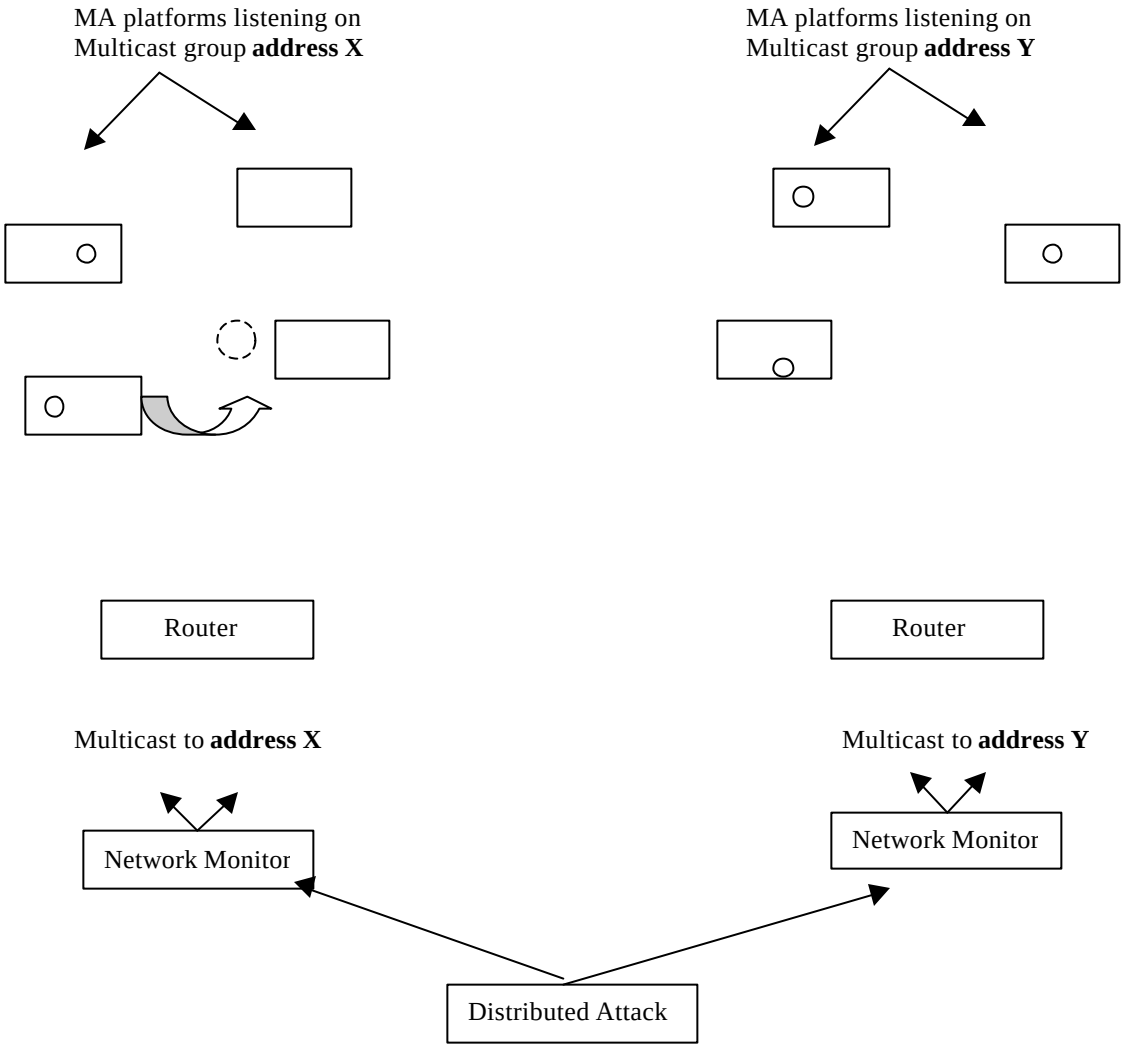

Fig 4. Agent Communication

4.4.3 Decoy Agents

Due to encryption an attacker will fail in his attempt to analyze traffic in order to determine the contents of the message as well as the nodes' locations. However the attacker could set few sniffing points across the network and analyze the data flow based on volume of traffic. Frequent and regular traffic flow towards certain points in the network might represent critical node activities.

By introducing *decoy agents* that roam around the network aimlessly generating garbage traffic in the network. This serves the purpose of diverting an attacker away from any critical system components. The garbage traffic inserted into the network by the decoy agents is a suitable mechanism to throw some confusion into the analysis process of the attacker.

4.4.4 Mobile Agent Security

The security of the mobile agents is one of the major concerns of our system development. The security of the agents can be compromised by other malicious agents or by a malicious platform hosting agents. When an agent arrives at it's platform, it runs under the control of the platform host. A malicious platform can destroy an agent or modify it's sequence of execution leading to the generation of incorrect results. The same could be caused by malicious agents. Either way, it is necessary for the higher level DIDS components to identify results that are generated by both malicious hosts and agents.

There exist an extensive research in this area with varying claims ranging from the impossibility of agent security to very complex solutions, like partial agent result encryption, obfuscated code, to provide agent security [Ng00]. We refrain from assuming that the agents are secured by some mechanisms. Even if the agents lack any means of securing themselves from malicious activities, the proposed intrusion detection scheme is better than the existing ones, since it enhances the security of the internal components by concepts of mobility and passivity.

Our approach to polling/voting takes advantage of redundant MA platforms in the network (Fig 5a). A group of agents is responsible for processing data multicast by the monitors. All the members of the agent group perform the same functions on the same data sets received from the monitors. Hence they are expected to produce the same results. These redundant results are sent to a voting server where a voting mechanism is carried out to determine the majority results (Fig 5b). This elected result is considered as correct result of the agents and is reported to the root node. Note that simple majority voting mechanisms can tolerate a certain number of messages lost in network traffic. Extensive research of voting mechanisms and their fault tolerance properties is out of the scope of this thesis, however, an interested reader can find some of the relevant information in [LKE89, SD89].

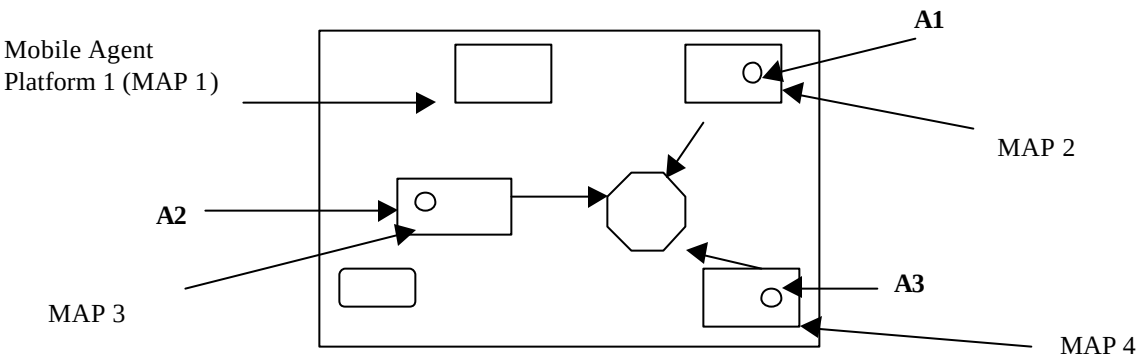

Fig 5a. Multiple agents submitting their results for Voting Server.

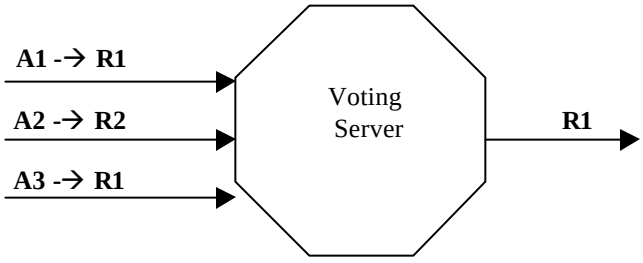

Fig 5b. Voting on the Voting Server.

Finally, our distributed intrusion detection scheme performs critical decision making tasks in a trusted host rather than an insecure MA platform. Whenever there is an incorrect result(s) or there are no results from agents, the voting server tags the agents and the MA platform as suspicious since the incorrect results may be due to either the malicious MA platform or the malicious agent or of both, as a worst case (Refer Appendix B for the algorithm that adjusts the trust levels of agents and hosts). The trust level of either the agent or an MA platform deteriorates when it is in disagreement with the others. At certain point in time, the voting server excludes considering the results of an untrusted agent or a MA platform (Fig 5c). In this case, a security breach alert is sent out to the system security officer (SSO). The voting server continues excluding the results until the SSO performs a post mortem analysis to eliminate any compromises made to the system.

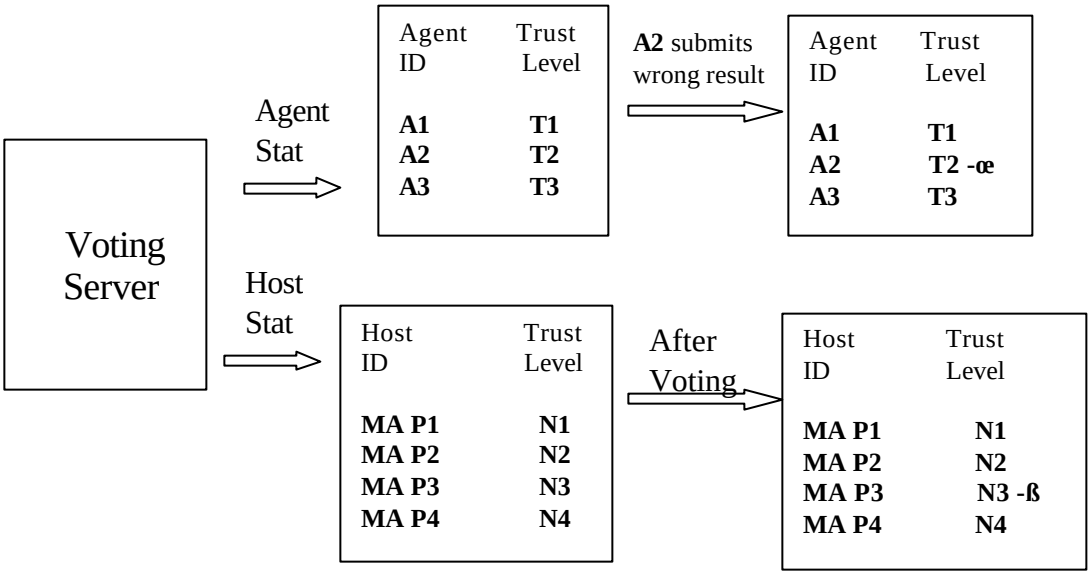

Fig 5c. Voting Server updating Agent/Host Trust Level.

4.4.5 Avoiding Malicious Host

The agents must avoid a malicious host of the MA platform. By avoiding these locations the agents could reduce their vulnerability to attacks aimed at destroying them or modifying their code to distract their intended functions. In order for the agents to avoid the malicious platforms they need to be informed about them. This is achieved by the voting server, which routinely distributes the list of malicious MA platforms. These messages are sent in the encrypted form, and the list of recipients excludes the hosts considered malicious. When an agent randomly chooses its next destination it avoids the malicious platform. As mentioned earlier, voting server adds an MA platform to its malicious host list when the host reaches the threshold indicative of a security breach.

5. System Implementation

This section provides information regarding the test environment, tools that were utilized for the system development and the detection mechanism that was used for DIDS's intrusion detection.

5.1 Intrusion Detection Technique

For the purpose of testing and to generate some tangible results, we implemented a simple version of the Graph based Intrusion Detection system (GrIDS) [CC96]. The GrIDS generates different shapes of graphs for a period of time that is an indication of a large scale distributed attack. The nodes and the links of the graph represent the compromised machines and the connection between the machines respectively. The further propagation of the attack to other machines leads the way to the growth of the tree. This tree representation is then summarized to produce results that are compared with threshold values for an indication of an attack. We walk through an attack scenario below.

Imagine an attacker 'A' trying to scan a class B (for instance 157.182.194/24) network for common vulnerabilities on ports 20, 21, 22 and 80. With the use a port scan tool he scans 10 machine for every 2 seconds with an interval of 3 seconds between each scan. After the first scan 'A' finds a compromised host in the network. He runs the expoit and gets root access. Now instead of running the scan from the first machine he runs the scan from the compromised machine in the network, this time scanning another subntet of the 16's network. Unless all these scans are traced back to the attacker 'A', concluding

that the attacker ‘A’ is responsible for a distributed attack would be impossible. At the beginning of the first scan the GrIDS begins to develop an attack tree for these activities as follows :

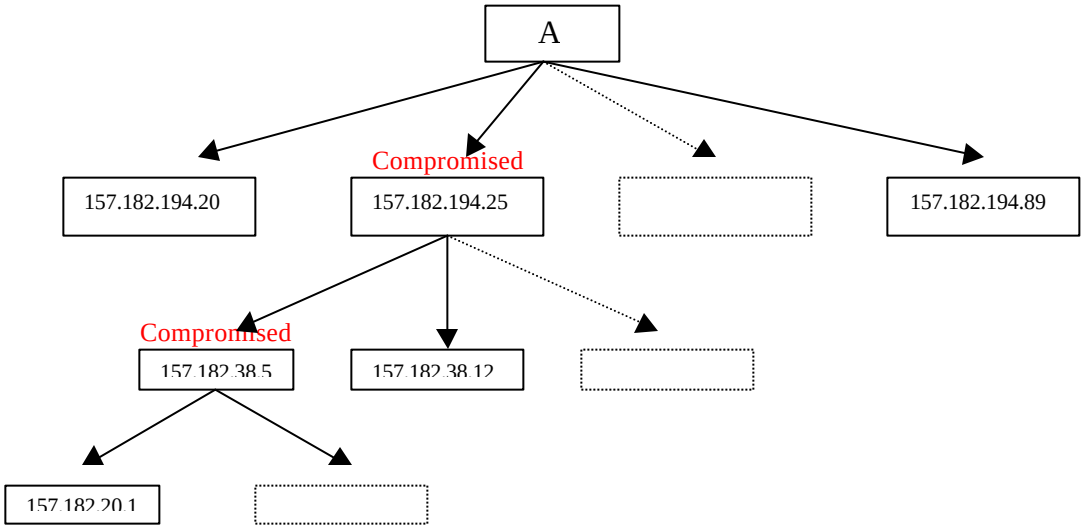

Fig 6. GrIDS tree for Scan Attack.

The description of the mechanism used to trace the different scans to the single attacker is beyond the scope of this document. A detailed design description is available in [CC99]. After the generation of the graph (Fig 6) GrIDS engine calculates a graph summary. In the above case it could be the number of machines and network that are scanned by the attacker ‘A’. If this exceeds threshold values GrIDS identifies the activity of ‘A’ as an attack.

In our system the mobile agents, as internal components of DIDS, are responsible for generating activity trees for large scans on the network that they supervise. The trees are generated when monitors (Leaf Nodes) alert them of any suspicious activities in their locality. The independent activity trees are then sent to the Root Node. The root node accumulates these independent graphs into one based on the relationship that exist

between them. This would lead to the graph summary and an alarm is raised in case of any attacks.

5.2 Mobile Agent Platform

Aglets Software Development Kit (ASDK) [Lan97,LO98], for the development of Java based mobile agents, was chosen due to its easy availability over the Internet. ASDK, developed by IBM, is a Java-based framework for implementing mobile agents. The Aglet runtime layer provides functionality for creating, dispatching and disposing a mobile agent. The Application Programmer Interface (API) is provided by a Graphical User Interface (GUI) that also provides an environment for hosting mobile agents (MA platforms). The Aglets server listens on a pre-specified port for any mobile agents that are to be hosted in its environment. Each Aglet has its own thread of execution. Hence the functionality of an Aglet is implemented in it's *run* method. Before an Aglet is dispatched to another Aglet server the class loader of the ASDK serializes the Aglets state of execution. This enables the Aglets to save their state information so they could be deserialized in another server.

5.3 Network Intrusion Detection System as Leaf Nodes

The leaf nodes of DIDS could have range of functionality varying from packet filtering to host based or network based intrusion detection. The leaf nodes are responsible for protecting their localities and report any anomalous activities to the internal nodes for further analysis. For our system we opted for *Snort* [Roe99], a light weight network intrusion detection system. *Snort* is capable of performing real-time

traffic analysis and logging on IP networks. Its misuse intrusion detection technique searches the traffic for any pattern matching with a set of predefined attack signatures. For our purpose we used *Snort* to detect any network wide port scans and log the details of the port scans. These alert logs are routinely sent as multicast transmissions to the agent for the generation of attack trees.

5.4 Scanning Tool

We used Nmap [NMAP] to simulate a scanning attack. Nmap is a freely available tool capable of carrying out rapid scans on a large network. It uses raw Internet Protocol (IP) packets to determine what hosts are available on the network, types of services provided by the hosts in a network and types of operating system that are used by the hosts in the network. Vanilla Transmission Control Protocol (TCP) connect scanning, TCP SYN scanning, TCP FIN scanning and Internet Control Management Protocol (ICMP) scanning are few of the type of scans that could be performed by the Nmap.

5.5 Test Bed

We carried out a simulation of our system. Due to the restrictions we had on the usage of the network, our test was carried on few machines located on a single sub network. The following figure illustrates the set up of our test architecture :

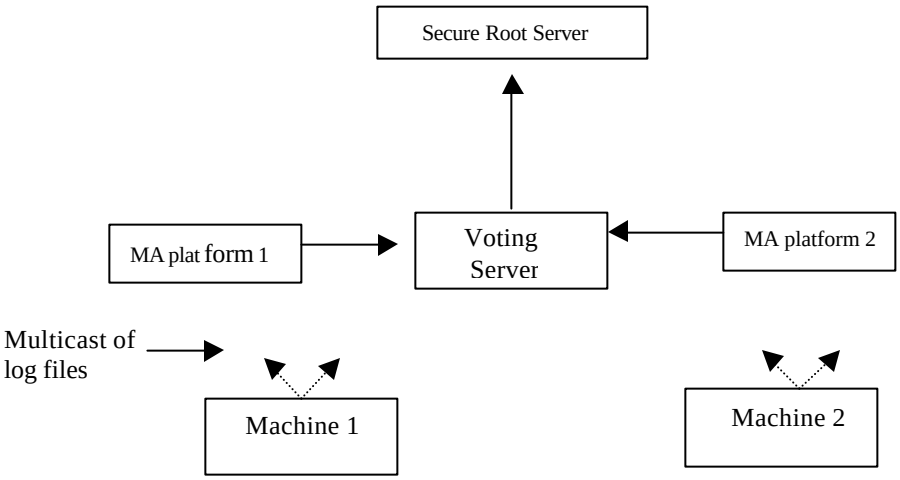

Fig 7. System test bed

Machine 1 and *Machine 2* serve as leaf nodes by running the Network Intrusion Detection Tool *Snort*. Any intrusions or intrusion attempts that are detected in these machines will be multicast to group addresses *mcasgrp 1* and *mcastgrp 2*, respectively. *MA platform 1* will listen for data transmissions on *mcastgrp1* and *MA platform 2* on *mcastgrp 2*.

We carried out a simple TCP connect() port scan on one of our Class B sub networks using Nmap's TCP half-open (SYN) scan option for few open ports on the hosts of the subnet. This scan was carried out as follows (Refer Appendix C for nmap options):

nmap – sp 20, 21, 22, 23, 80, 111 157.182.194.89 157.182.194.39

A TCP half-open scan sends a SYN packet to a port on the host and waits for a reply. An open port would send would send back a SYN/ACK packet on the other hand a closed port sends a RST packet. If a SYN/ACK packet is received a RST packet is sent to tear down the connection. Usually a host logs details of a complete TCP connection initiation. The TCP half-open scan is less likely to be logged since the connection was not complete. The execution of this Nmap command produced the following result :

Starting nmap V. 2.53 by fyodor@insecure.org (www.insecure.org/nmap/)

Interesting ports on localhost (157.182.194.89):

(The 1 port scanned but not shown below is in state: closed)

Port	***State***	***Service***
21/tcp	***open***	***ftp***
22/tcp	***open***	***ssh***
23/tcp	***open***	***telnet***
80/tcp	***open***	***http***
111/tcp	***open***	***sunrpc***

Interesting ports on daffy.csee.wvu.edu (157.182.194.39):

(The 3 ports scanned but not shown below are in state: closed)

Port	***State***	***Service***
21/tcp	***open***	***ftp***
80/tcp	***open***	***http***
111/tcp	***open***	***sunrpc***

Nmap run completed -- 2 IP addresses (2 hosts up) scanned in 0 seconds

The following scans on the two machines were detected and logged by *Snort*. The log files look as follows :

157.182.194.89's Port_Scan.log

Jul 4 14:53:54 157.182.194.28:63681 -> 157.182.194.89:20 SYN ******S*

Jul 4 14:53:54 157.182.194.28:63682 -> 157.182.194.89:21 SYN ******S*

Jul 4 14:53:54 157.182.194.28:63683 -> 157.182.194.89:22 SYN ******S*

Jul 4 14:53:54 157.182.194.28:63684 -> 157.182.194.89:23 SYN ******S*

Jul 4 14:53:54 157.182.194.28:63685 -> 157.182.194.89:80 SYN ******S*

Jul 4 14:53:54 157.182.194.28:63686 -> 157.182.194.89:111 SYN ******S*

157.182.194.39's Port_Scan.log

Jul 4 14:53:54 157.182.194.28:63681 -> 157.182.194.39:20 SYN ******S*

Jul 4 14:53:54 157.182.194.28:63682 -> 157.182.194.39:21 SYN ******S*

Jul 4 14:53:54 157.182.194.28:63683 -> 157.182.194.39:22 SYN ******S*

Jul 4 14:53:54 157.182.194.28:63684 -> 157.182.194.39:23 SYN ******S*

Jul 4 14:53:54 157.182.194.28:63685 -> 157.182.194.39:80 SYN ******S*

Jul 4 14:53:54 157.182.194.28:63686 -> 157.182.194.39:111 SYN ******S*

In the rest of this section we will refer the computer with IP address **157.182.194.39** and the computer with IP address **157.182.194.89** as *Machine 1* and *Machine 2* respectively. A script that runs on the these two machines routinely transfers

the log files to the *MA*, through multicasting, in the following format (*Machine 1* multicasts it's log files to *Mcast grp 1* and *Macine 2* to *Mcast grp 2*) :

157.182.194.89's Transfer Format

157.182.194.28 157.182.194.89:20, 157.182.194.89:21, 157.182.194.89:22

157.182.194.89:23, 157.182.194.89:80, 157.182.194.89:111

157.182.194.39's Transfer Format

157.182.194.28 157.182.194.39:20, 157.182.194.39:21, 157.182.194.39:22

157.182.194.39:23, 157.182.194.39:80, 157.182.194.39:111

In the above format the attacker machine's IP address is followed by a list of comma separated **IPaddress:port** entries, which are the ports the attacker scanned on a machine. The MA platforms listening on the multicast group receive and store these log data for agent processing. As we mentioned earlier, the mobile agents build a tree representing attacks on their locality based on these log file data.

Mobile agents belonging to *MA platform 1* built a tree as follows :

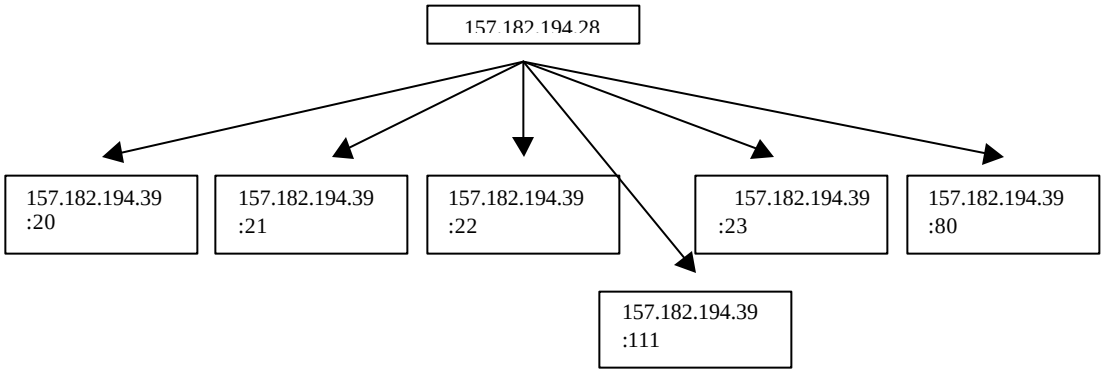

Fig 8. Attack Tree of MA platform 1

Mobile agents belonging to *MA platform 2* build a tree as follows :

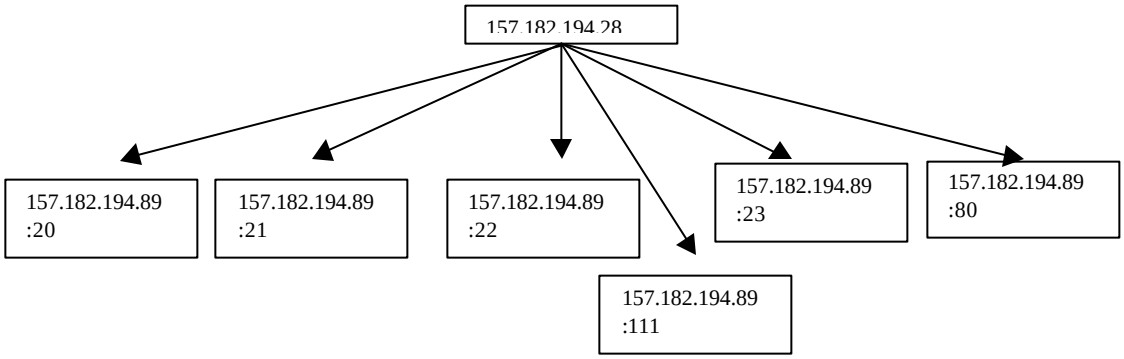

Fig 9. Attack Tree of MA platform 2

Redundant generation of these attack trees by various agents will be submitted to the voting server for voting. The voting server picks the majority result and submits it to the *Root Node*. The *Root Node* merges these two sub graphs together, since the source responsible for this suspicious activity is the same. The *Root Node* maintains this graph for a defined period of time. If the graph grows above the size of the threshold the *Root Node* concludes the activity as a large scale scan and raises the alarm.

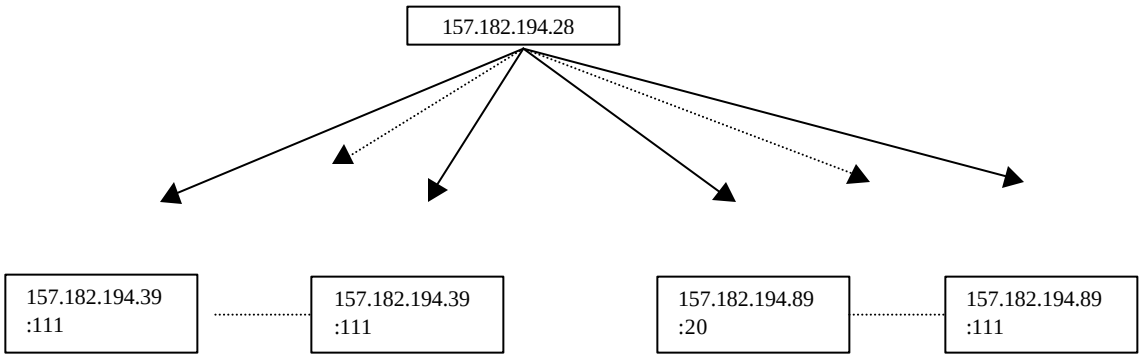

Fig 10. Root node's generation of an attack tree

A typical calculation may involve counting the number of machines and the ports that were scanned by the attacker and comparing it to see whether a predefined threshold has been reached.

5.6 Discussion

Our system was entirely designed using the Java language. The choice of Java was mainly due to the association between Aglets and Java (Aglet is a Java-based mobile agent). Windows NT platforms were used to run the Aglets servers. The Linux based machines were used to run the Network Intrusion Detection tool, *Snort*, and were responsible for monitoring functions. One of the major problem we faced during our implementation phase was due to the fact that the ASDK was itself in a development stage. Certain features that were mentioned in the ASDK documentation did not exist on the implementation that was provided. As a result we had to adopt different techniques to work around the problems. This section discusses some of the issues that arose in the development phase.

In the current implementation, the leaf nodes multicast their data to the MA platform which maintains this data in a standard configuration directory. When agents arrive to an MA platform they poll this directory for the availability of new files. After processing, these old files are moved to a back up directory. Multicasting the data directly to the agents is another form of data transmission between agents and the leaf nodes. Our choice for the first method was due to the inability of the current implementation of the Aglet framework to maintain references to remote Aglets that have been created and dispatched to other locations. Therefore an agent cannot retrieve any data transmitted during it's movement. Unless a reliable mechanism is put in place for ensuring the agents receive all the data once they arrive in a location, this method allows for large data loss due to agent's constant movement.

Reliability of multicast data remains a concern in both techniques. We could easily utilize a reliable multicast protocol [MR98] to provide a degree of reliability to the data transmission. However, as we discussed earlier, the requirement for reliability should be evaluated on a case by case basis.

Another problem that stems from the Aglets framework's inability to make references to the remotes objects is as follows, when an agent is tagged as malicious the security control node is not able to destroy the agent. Although our implementation disregards any results provided by malicious agents, the agent will still exist in the network. All the MA platforms could be informed of malicious agents so that they could destroy the agent on the arrival to their platform. However providing MA platforms the ability to destroy agents could lead to a security breach of the system. A safer solution

would be to let the MA platform inform the security control node about the arrival of the malicious agent. Until the security control node retrieves the agent the platform should prevent any attempts by the agent to mobilize to another location.

Currently, our implementation uses simple TCP oriented mechanism to send and receive node control information between the components. For instance the distribution of malicious host list by the voting server to the MA platforms is carried out through a TCP based connection. Instead, a Simple Network Management Protocol (SNMP) [CF90] could be used. SNMP is a protocol designed to facilitate the exchange of management information among network devises. SNMP consists of *agents* that run on managed devices and a *manager* that runs on the device responsible for managing others. The *manager* sets and queries control information through *Management Information Base (MIB),* a database that specifies variables maintained by the *agents.* Since many existing networks utilize SNMP for management purposes, the same could be used for our system with a least number of modifications.

In order to dispatch an agent to a random location, we indexed all MA platforms, generated a random value between zero and the total number of available MA platforms and picked the MA platform that holds the random number as it's index. Smaller the range higher the possibility for a 'collision'. A collision we refer to more than one operational agent being allocated to a MA platform at a time. A light collision would not have a major impact on the MA platform's host performance, since these agents are threads of execution (light weight processes). In a widely distributed environment it is possible to have a large number of MA platforms to reduce frequent and heavy collisions.

The voting server requires the timely submission of processed results by all the agents in a single session. Due to high collision rate in MA platforms, congestion on the local network and an MA platform's lower processing speed could cause an agent to submit it's results to the voting server later than the other agents in the same domain. A significant delay, due to the above reasons, could improperly lead the voting server to reduce the trust level of these agents. This could propagate to the point at which non-malicious agents are being destroyed. In our implementation the voting server tolerates this delay by waiting for constant time period after the first submission of a result by any of the agent. This is clearly an incomplete solution for the problem since the delay is not a constant factor. Refer the 'Future Work' section of this document for more discussions on solutions to this problem. An interested reader could refer to TCP protocol's Round Trip Time (RTT) calculation. TCP has to incorporate congestion on the network before it determines to resend packets that are considered to be lost.

6. Conclusion and Future Work

Due to the rapid growth of the Internet there have been large a deployment of computer networks over the last decade. The successful inter-operation of these networks largely depends on their capabilities to communicate between each other. This has resulted in a well defined communication infrastructure where almost any single system can communicate with another system over the network. Further the delegation of large and critical information processing to the computers has increased their importance among our human society. At the same time increasing their vulnerability to attacks. Hence it has become very critical to protect computer networks from malicious attackers who try to interrupt the functionality of these computers.

Intrusion Detection Systems (IDS) have become an important component in the security architecture of a computer. It forms the last line of defense against the attackers. Detecting intrusions in a distributed environment requires the components of IDS to spread across different network and gather and process information in different network localities. As a result, the security of these components itself becomes a great concern. Protecting these components is necessary for the proper functioning of the Distributed Intrusion Detection System. In this research we proposed and implemented an attack resistant architecture using DIDS components as mobile agents.

We plan to develop an API for our mobile agent based internal component so that it could be incorporated into any hierarchical intrusion detection system and provide attack resistance capability. Developers could conveniently use our API to customize and extend the mobile agent architecture to their personal needs.

Further improvements include utilization of specialized inter-agent communication frameworks. In the current implementation, the information passing between the nodes in the IDS hierarchy is carried out through a data structure that holds information necessary for detecting an intrusion. In the future we would like to use standard agent communication languages, KQML for instance. This will provide interoperability between multiple Commercial Off the Shelf (COTS) intrusion detection systems that might be utilized in a large distributed environment. This modification would also provide a better framework for information sharing between agents and interoperability of various intrusion detection systems.

Our system still contains several single points of failure. These are, for example, the voting server and the root node of the IDS. Elimination of these functionally critical failure points is a major challenge. Making them mobile as the rest of the components could be a way of solving the problem. However, it is not clear how would this affect the performance of the overall system.

References:

[AK98] R. Atkinson, S. Kent. Security Architecture for the Internet Protocol, *RFC 793*, Information Sciences Institute, University of Southern California, CA, USA, November 1998.

[AL95] Debra Anderson, Teresa F. Lunt, et.al. Detecting Unusual Program behavior Using the Statistical Component of the Next-generation Intrusion Detection Expert System. *Report SRI-CSL-95-06*, Computer Science Laboratory, SRI International, May 1995.

[And80] J.P Anderson. Computer Security Threat Monitoring and surveillance. *Technical Report*, James P Anderson Co., Fort Washington, Pennsylvania, 1980.

[Axe99] Stefan Axelsson. Intrusion Detection Systems: A Taxonomy and Survey. *Technical Report No 99-15,* Dept. of Computer Engineering, Chalmers University of Technology, Sweden, March 2000.

[BG98] Jai Sundar Balasubramaniyan, Jose Omar Garcia-Fernandez, et.al. An Architecture for Intrusion Detection using Autonomous Agents. *Technical Report 98/05,* COAST Laboratory, Purdue University, IN, USA, June 1998.

[CB94] William R. Cheswick and Steven M. Bellovin. *Firewall and Internet Security.* Addison Wesley, First Edition, 1994.

[CC96] S.Stainford-Chen, Steven Cheung, et.al. GrIDS-Graph Based Intrusion Detection System for Large Networks. In the Proceedings of the *19th National Information Systems Security Conference,* Baltimore, MD, October 1996.

[CC99] Steven Cheung, Rick Crawford, et.al. The Design of GrIDS : A Graph Based Intrusion Detection System. Department of Computer Science, University of California at Davis, CA, USA, January 1999.

[GS96] Simson Garfinkel and Gene Spafford. *Practical Unix and Internet Security.* O'Reilly & Associates, Second Edition, 1996.

[Hak99] Hakan Kvarnstrom. A Survey of Commercial tools for Intrusion Detection. T*echnical Report 99-8,* Dept. of Computer Engineering, Chalmers University of Technology, Sweden

[ISS01] Interner Security Systems (ISS), RealSecure. Available at http://www.iss.net/securing_e-business/security_products/intrusion_detection.

[Lan97] Danny B. Lange. Java Aglet Application Programming Interface (J-AAPI), *White Paper - Draft 2*. Available at http://www.trl.ibm.com/aglets/JAAPI-whitepaper.html. 1997.

[LKE89] P. R. Lorczak, A. K. Koglayan, D. E. Eckhardt, "A Theoretical Investigation of Generalized Voters for Redundant Systems,", FTCS 19, Digest of papers, 1989.

[LO98] Danny B. Lange, M. Oshima, *Programming and Developing Java Mobile Agents with Aglets*, Addison-Wesley, Menlo Park, CA, 1998.

[MHL94] Biswanath Mukherjee, Todd L. Heberlein, Karl N. Levitt. Network Intrusion Detection. *IEEE Network*, 8(3):26-41, May/June 1994.

[MR98] A. Mankin, A.Romanw, et.al., IETF Criteria for Evaluating Reliable Multicast Transport and Application Protocols, *RFC2357*, Information Sciences Institute, University of Southern California, CA, USA, June 1998.

[Ng00] Ng, Sau-Koon. Protecting Mobile Agents against Malicious Hosts, *Master Thesis*. Division of Information Engineering, The Chinese University of Hong Kong, June 2000.

[PM99] Peter Mell and Mark McLarnon. Mobile Agent Attack Resistant Distributed Hierarchical Intrusion Detection Systems. In the *Proceedings of the 2nd International Workshops on Recent Advances in Intrusion Detection*, West Layfayette, Indiana, USA, 1999.

[Pos80] Postel, J. (ed.), User Datagram Protocol, *RFC 768*, Information Sciences Institute, University of Southern California, CA, USA, August 1980.

[Pos81] Postel, J. (ed.), Internet Protocol - DARPA Internet Program Protocol Specification, *RFC 791*, Information Sciences Institute, University of Southern California, CA, USA, September 1981.

[Pos81] Postel, J. (ed.), Transmission Control Protocol - DARPA Internet Program Protocol Specification, *RFC 793*, Information Sciences Institute, University of Southern California, CA, USA, September 1981.

[Roe99] Martin Roesch. Snort – Lightweight Intrusion Detection for Networks. In the *Proceedings of the USENIX LISA '99 Conference*, Seattle, WA, USA, 1999. Available at http://www.snort.org.

[San95] Sandeep Kumar. Classification and Detection of Computer Intrusions. *Ph.D Thesis*, Purdue University, IN, USA, August 1995. Available at https://www.cerias.purdue.edu/techreports-ssl/public/95-08.pdf.

[SD89] K. G. Shin, J. W. Dolter, "Alternative Majority-Voting Methods for Real-Time Computing Systems,", IEEE Transactions on Reliability, V. 38, No. 1, April 1989.

[Ver88] Vernon Paxon. Bro: A system for detecting network intruders in real time. In the Proceedings of the *7th USENIX Security Symposium*, San Antonio, TX, USA, 1998.

Appendix A

Aglet Framework

Aglets are Java-based mobile agents developed by the IBM, Japan research lab. The Aglet Development Tool Kit (ASDK) provides a set of packages (com.ibm.aglet) that are necessary for the development of mobile agents. In order for the execution of an Agent, a system should run an "Aglet host" that provides the execution environment for the Aglet. The Aglet runs as a thread inside the context of the host application.

An Aglet could experience the following stages throughout its life cycle :

Creation : A new agent is created and initialized. After the creation the Aglet begins executing it's *run* method.

Cloning : Creating a exact copy of an existing Aglet. All the state information is copied.

Dispatch : Mobilizing an Aglet to a different location. During this stage all non transient and non static variables' states are saved.

Retract : Acquiring a previously dispatched Aglet.

Deactivate : The execution of the Aglet is halted. The Aglet goes to sleep until it's Activate again.

Activate : Waking up a deactivated Aglet.

Dispose : Destroying an Aglet.

The following figure illustrates the *AgletFramework* :

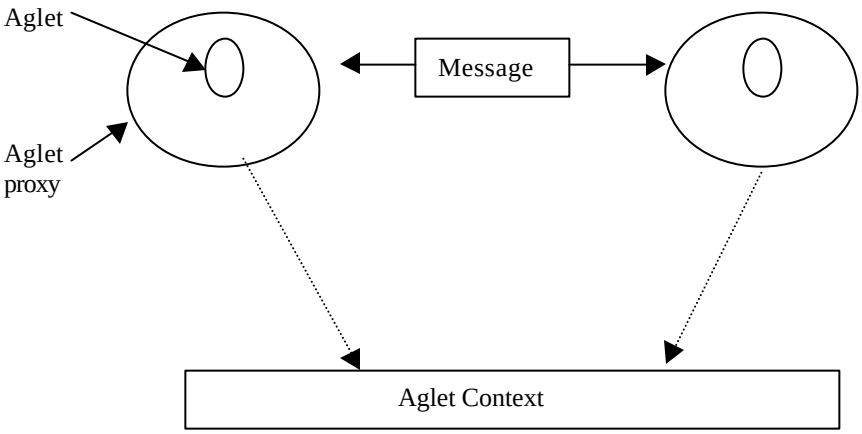

Fig 11. Aglet Framework

The *Aglet Context* provides the environment for the creation and the execution of Aglets (Framework). The *Aglet Proxy* serves as an handle for handling communication between agents. Please note as we mentioned earlier the current implementation of the Aglet Framework lack the ability to maintain a reference to a remote Aglet. Hence when an Aglet is created and dispatched from a framework, it loses all the references to that Aglet.

Examples of Aglet Creation

The following example provides a simple example on how to write Aglets. The class attempting to create an Aglet should subclass class `Aglet`, which includes several methods that can be overridden to customize the behavior of the Aglet.

```
import com.ibm.aglet.*;
public class AnAgent extends Aglet {
```

`The onCreation()` method can be overridden to initialize an Aglet. The `onCreation()` method is invoked only once in an Aglet's lifetime and should be used

only for initialization.

```
public void onCreation(Object init) {
        //Operations performed at the time of Aglet creation
}
```

The Aglet also has a `run()` method, which represents the entry point for the Aglet's main thread. This is similar to the `main()` method of a Java application, except that `run()` is invoked each time an Aglet arrives at a new Aglet host.

```
public void run() {
        //Activity of the thread
}
```

Each time the `Aglet` arrives at a new host, a method called `onArrival()` would be invoked to perform any initialization. Once `onArrival()` completes, `run()` would be invoked to get the Aglet started again at the new host.

```
public void onArrival() {
        // Operations performed at time of arrival
}
```

Before an Aglet is dispatched to another location the, it's `onDispatch()` is invoked. This method enables the Aglet to perform any cleanup operations before it is serialized. When it returns from `onDispatch()`, its state will be serialized and all its threads terminated. The class files and serialized state will then be sent to the new host, where the Aglet will be resurrected.

```
public void onDispatching() {
        Operations performed before being dispatched
}
```

Our Say on Aglet Security

The Aglet Framework provides mechanisms to prevent attacks on the host by the agents. This is achieved through a similar architecture as the JDK security architecture, where a *Policy File* defines the permissions available for the Aglets. Each permission specifies a permitted access to a particular resource, such as read and write access to a specified file or directory or connect access to a given host and port.

However their are no measures till now for securing Aglets being attacked by malicious hosts. Hence deploying Aglets as mobile agents for a real world problem would exhibit various insecure features. Unless additional mechanisms are put in place.

Appendix B

The voting server holds the responsibility for determining the valid voting candidates. In order to arrive at this decision, the voting server maintains certain pointers, in the form of objects, that are indicative of mobile agents and MA platforms trust to operate as intended.

The agent's trust level object consists of four values:

number of correct result submissions

number of wrong result submissions

number of no result submissions

A user defined initial agent trust value

The MA platform's trust level consists of a value representing number of no result submissions and a user defined initial platform trust value.

Each time a voting takes place the agents' and platforms' quantitative values are updated. After certain number of agent result submissions the trust vales are updated based on frequency of wrong result submission and no result submission. The following algorithm illustrates the mobile agents' and hosts' trust level updates:

Assume that *max_entry* holds the majority result after voting is been carried out for a single event generated by the monitor.

```
For each agent ID  {
    if (agent_result (agent ID)= max_entry) then
      AgentStat (agent ID).submit += 1;
    else {
      if agent_result (agent ID) = 0 then
            AgentStat (agent ID).nosubmit += 1;
      else
```

```
                AgentStat (agent ID).wrong_result += 1;
          HostStat(agent ID).noresults += 1;
        }
 }
if (perform_analysis) {
        total = submit + nosubmit + result;
     nosubmit_fre = nosubmit / total;
        result_fre = result / total;
        if (nosubmit_fre >= nosubmit_threshold)
          AgentStat (agent ID). reduceTrust(1);
        if (result_fre >= result_threshold)
           AgentStat (agent ID). reduceTrust(2);
        if (AgentStat (agent ID). getTrust ()<= trust_threshold)
          delete(Agent(agent ID));
    else update(AgentStat(agent ID);
        if (HostStat(agent ID).noresults / total > host_trust_threshold)
          delete(HostStat(agentID);
        else update(HostStat(agentID);
}
```

Appendix C

1. Aglets Daemon

SYNOPSIS

agletsd [-options]

DESCRIPTION

agletsd is batch file that sets the necessary environment variables and starts the Aglet Server.

Options

-help	print out this message.
-verbose	turn on verbose mode.
-port <port>	set the port used by daemon.
-viewer <class>	set the viewer class.
-resolve	use fully qualified hostname by resolving reverse lookup.
-nosecurity	disable security manager.
-commandline	use command line interface.
-nogui	no gui/cui.
-domain	set the domain name.

2. Snort Tool

SYNOPSIS

snort [-abCdDeINopqsvVxX?] [-A *alert-mode*] [-c *rules-file*] [-F *bpf-file*] [-g *grpname*] [-h *home-net*] [-i *inter- face*] [-l *log-dir*] [-L *bin-log-file*] [-M *smb-hosts-file*] [-n *packet-count*] [-r *tcpdump-file*] [-S *n=v*] [-t *chroot_directory*] [-u *usrname*] *expression*

DESCRIPTION

Snort is an open source network intrusion detection sys- tem, capable of performing real-time traffic analysis and packet logging on IP networks. It can perform protocol analysis, content searching/matching and can be used to detect a variety of attacks and probes, such as buffer overflows, stealth port scans, CGI attacks, SMB probes, OS fingerprinting attempts, and much more.

Options

-A alert-mode Alert using the specified *alert-mode.* Valid alert modes include **fast, full, none,** and **unsock.**

-a Display ARP packets when decoding packets.

-b Log packets in a **tcpdump(1)** formatted file.

-c config-file

-C Print the character data from the packet payload only (no hex).

-d Dump the application layer data when displaying packets in verbose or packet logging mode.
-D Run Snort in daemon mode. Alerts are sent to /var/log/snort/alert unless otherwise specified.
-e Display/log the link layer packet headers.
-F bpf-file Read BPF filters from bpf-file.
-g <**grpname**> Change the GID Snort runs under to <grpname>
-h home-net Set the "home network" to *home-net.*
-i interface Sniff packets on *interface.*
-I Print out the receiving interface name in alerts.
-l log-dir Set the output logging directory to *log-dir.*
-L binary-log-file Set the filename of the binary log file to *binary- log-file.*
-M smb-hosts-file Send WinPopup messages to the list of workstations contained in the *smb-hosts-file .*
-n packet-count Process *packet-count* packets and exit.
-N Turn off packet logging. The program still gener- ates alerts normally.
-o Change the order in which the rules are applied to packets.
-O Obfuscate the IP addresses when in ASCII packet dump mode.
-p Turn off promiscuous mode sniffing.
-q Quiet operation. Don't display banner and initial- ization information.
-r tcpdump-file Read the tcpdump-formatted file *tcpdump-file.*
-s Send alert messages to syslog.
-S n=v Set variable name "n" to value "v".
-t chroot Changes Snort's root directory to *chroot*
-u uname Change the UID Snort runs under to *uname*
-v Be verbose.
-V Show the version number and exit.
-X Dump the raw packet data starting at the link layer.
-? Show the program usage statement and exit.
expression selects which packets will be dumped.
(The expressions are similar to the TCPDUMP expressions).

3. Nmap Tool Options

SYNOPSIS

nmap [Scan Type(s)] [Options] <host or net #1 ... [#N]>

DESCRIPTION

Nmap is designed to allow system administrators and curi- ous individuals to scan large networks to determine which hosts are up and what services they are offering. *nmap* supports a large number of scanning techniques such as: UDP, TCP connect(), TCP SYN (half open), ftp proxy (bounce attack), Reverse-ident, ICMP (ping sweep), FIN, ACK sweep, Xmas Tree, SYN sweep, IP Protocol, and Null scan.

OPTIONS

-sT TCP connect() scan
-sS TCP SYN scan
-sF -sX -sN Stealth FIN, Xmas Tree, or Null scan modes
-sP Ping scanning
-sU UDP scans

-sO	IP protocol scans
-sA	ACK scan
-sW	Window scan
-sR	RPC scan
-b <ftp relay host>	FTP bounce attack
-p <port ranges>	This option specifies what ports you want to specify.
-e <interface>	Tells nmap what interface to send and receive packets on.
-g <portnumber>	Sets the source port number used in scans.
-n	Tells Nmap to **NEVER** do reverse DNS resolution on the active IP addresses it finds.
-R	Tells Nmap to **ALWAYS** do reverse DNS resolution on the target IP addresses.
-r	Tells Nmap **NOT** to randomize the order in which ports are scanned.