T00924

Intrusion Detection and Response using Mobile Agent Technology

Developed by

Muhammad Ashraf Nadeem

Reg.# 25-CS/MS-01

Supervised by

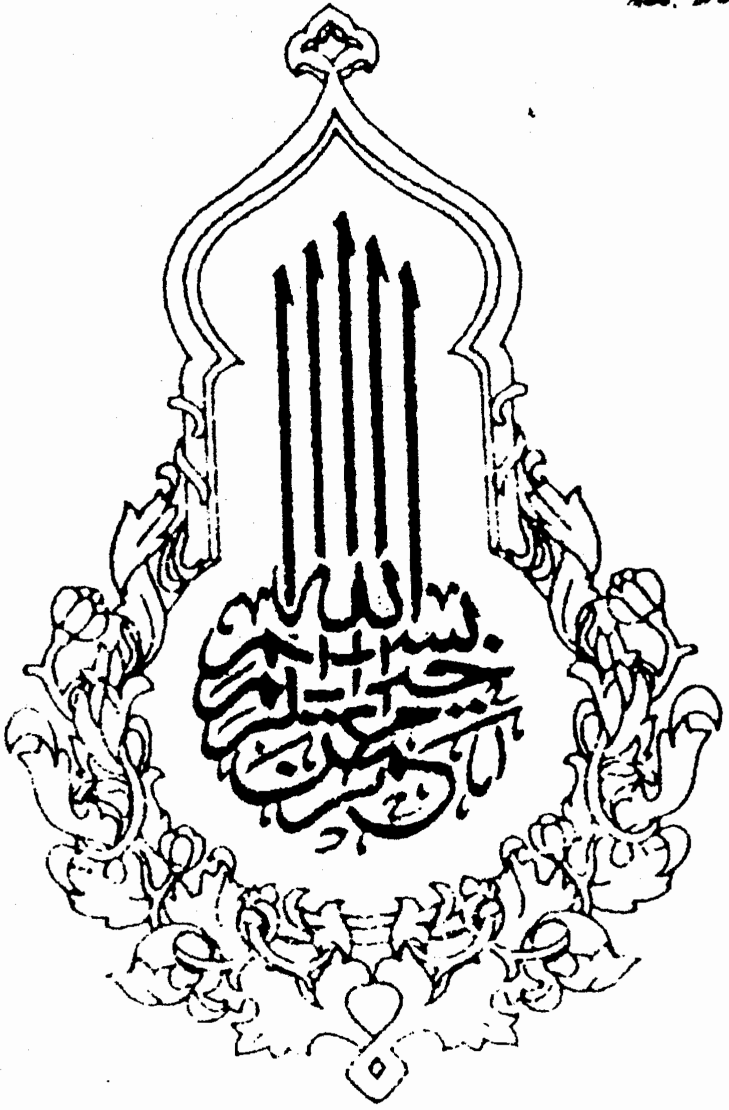
Dr. Muhammad Sher

Department of Computer Science
International Islamic University, Islamabad
(2004)

MS
005.8
NAI

1- computer security.
2- computer networks - Security measures.

Acc. No. (FMS) T-924



Allah's Name I Begin With, Most Compassionate, Most Merciful

International Islamic University, Islamabad

Dated: <u>21 February 2004</u>

Final Approval

It is certified that we have read the project report, titled "Intrusion Detection and Response using Mobile Agent Technology" submitted by Muhammad Ashraf Nadeem. It is our judgment that this project is of sufficient standard to warrant its acceptance by the International Islamic University, Islamabad, for the Degree **M.S. Computer Science**.

<u>Committee</u>

External Examiner
Dr Nazir A. Sangi,
Head,
Department of Computer Science,
AIOU, Islamabad.

[illegible]

Internal Examiner
Dr. S. Tauseef-ur-Rehman,
Head,
Department of Telecommunication Engg. Sciences,
IIUI

[illegible]

Supervisor
Dr. Muhammad Sher
Department of Computer Sciences
International Islamic University,
Islamabad.

[illegible]
21/02/04

Islamic University Faculty of Management Sciences

A dissertation submitted to the
Department of Computer Science,
International Islamic University, Islamabad
as a partial fulfillment of the requirements
for the award of the degree
MS Computer Science

Declaration

I hereby declare that this software, neither as a whole nor as a part thereof has been copied out from any source. It is further declared that I have developed this software entirely on the basis of personal efforts made under the sincere guidance of my teachers. No portion of the work presented in this report has been submitted in support of any application for any other degree or qualification of this or any other university or institute of learning.

Muhammad Ashraf Nadeem

Dedication

Dedicated to **The Holy Prophet Muhammad** (Allah's grace and peace be upon him) lord of the world and the thereafter. I offer my humblest thanks to him, who made us aware of our creator and guided us to the track, which leads to the success, who is a symbol of love and affection for all the creatures of Allah.

Acknowledgements

First of all I pay my humble thanks to **Almighty Allah** who knows all the things hidden or evident in this universe, even the things which pass through our hearts, who created me and gave me the courage to complete this project. It is said in the Holly Quran:

"Does man think that he will be left uncontrolled, (without purpose)? Was he not once a drop of ejected semen? Then he became a clot, so He created and fashioned him and made him into two sexes, male and female. Is He who does this, not able to bring the dead to life?" [Surah alQiyama: 36-40]

Without His help and blessings, I was unable to complete the project.

I also offer my humblest thanks to **The Holy Prophet Muhammad** (Allah's grace and peace be upon him), who made us aware of our creator and guided us to the track, which leads to the success, who is a symbol of love and affection for all the creatures of Allah.

I feel highly privileged in taking opportunity to express my profound gratitude and sense of devotion to my supervisor **Dr. Muhammad Sher,** DCS, International Islamic University, Islamabad for his inspiring guidance, consistent encouragement, sympathetic attitude, and dynamic supervision. Special thank to **Dr.Hafiz Farooq Ahmed(NIIT)** and **Dr. Muhammad Saleem Rathore (PAEC)**who gave me their precious time and helped me a lot to figure out the problems and to get rid of confusions.

I cordially regard the inspiration, prays, encouragement and all kinds of support of my loving and affectionate parents and family for their motivation in every aspect of my study enabling me to complete this project.

I also express my cordial and humble thanks to all my friends and teachers for their untiring help and cooperation in completing this project.

Muhammad Ashraf Nadeem

Project in Brief

Project Title:	Intrusion Detection and Response using Mobile AgentTechnology
Objective:	To detect and respond the intruders in an automated way at computer speed using mobile agent technology.
Undertaken By:	Muhammad Ashraf Nadeem
Supervised By:	Dr. Muhammad Sher
Technologies Used:	• TCL/TK ver 7.0 • D' Agents • Java
System Used:	Intel® Pentium III
Operating Systems Used:	• Microsoft® Windows® 2000 Professional • Red hat Linux(family) • Microsoft® Windows® XP Professional
Date Started:	27th November, 2002
Date Completed	30th October, 2003

Abstract

Today the computer security community is in search of novel solutions to achieve efficient detection and response mechanisms. It is especially because attackers intervene in an automated way, at computer speed. Therefore, there is need of such intrusion detection and response systems, which detect and respond at the same speed so that damage may be minimized.

We have designed an intrusion detection and response system prototype based on mobile agents. Our agents travel between systems in a network, obtain information, classify & correlate the information and report to the manager of the intrusion detection and response system (IDRS) which is responsible for responding the attack.

Table of contents

Chapter 1

Introduction

1 Introduction

Behind every invention or creation, there exists a need. The need behind the invention of aero planes was to allow people to travel faster. The need behind the invention of electric light was to allow better visibility during dark hours. Still, in this age, many things are being invented based on certain human needs.

Software engineering works on the same principles. A new human need in the world of computers gives birth to a new computer system every time. Hence, our system is based upon strong human need as well.

1.1 Background

The concept of intrusion detection was first proposed by James Anderson [1] in 1980, did not blossom until 1987 when Dorothy Denning published her seminal intrusion detection model [2]. Early IDS implementations employed a monolithic architecture whereby data collected at a single host was analyzed at a central point, at or adjacent to the point of collection [3, 4, 5]. Because monitoring account activity on a single host does not reveal attacks involving multiple hosts, IDS designers subsequently developed network-based IDSs that use a model of the network traffic to infer anomalies or misuses from low-level network packets traveling among hosts [6]. Network-based IDSs can be characterized as a change in perspective from host-centric to network-centric detection. A network-centric approach resolves a number of performance and integrity problems as well as problems associated with the reliance on audit trails. [7].

The US government provided significant funding for research in IDSs realizing that its computer systems were insecure. Hundred of millions of dollars have probably been spent on IDS research within last fifteen years [11].

Nearly all present-day commercial IDSs follow a hierarchical architecture. Information gathering occurs at leaf nodes, network-based or host-based collection points. Event information is passed to internal nodes that aggregate information from multiple leaf nodes. Further aggregation, abstraction, and data reduction can occur at higher internal nodes until the root node is reached. The root is a command and control system that evaluates attack situations and issues responses. The root typically reports to an operator console where an administrator can manually assess status and issue commands.

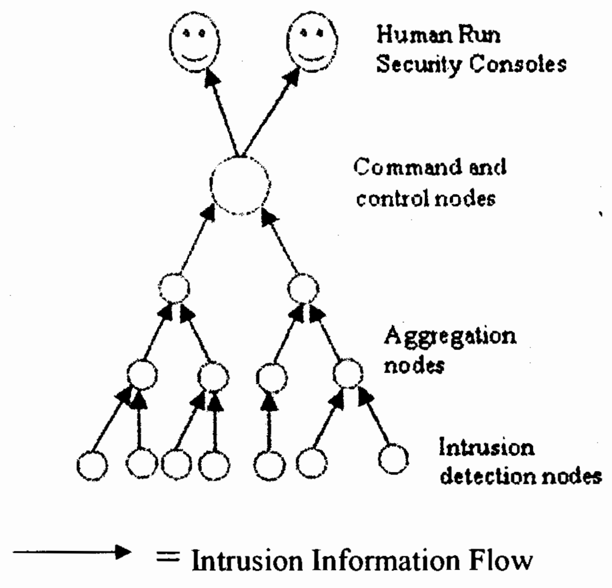


Figure 1.1Distributed Hierarchical Intrusion Detection Architecture

In general, hierarchical structures result in efficient communications, whereby refined information filters upward in the hierarchy and control downward. The architecture is excellent for creating scalable distributed IDSs with central points of administration, but somewhat rigid because of the tight binding between functionality and lines of communication that tend to evolve. While IDS components tend implicitly toward a hierarchy, this tendency is not strict. Communications can occur, in general, between any type of components and not solely on a one-to-one or master/slave basis. For example, to improve notification and response, a collection unit may directly communicate a critical event to the command and control node, as well as an aggregation node. Moreover, peer relationships among command and control nodes are needed when different administrations manage portions of an enterprise network, or distinct and separate networks [8].

At least one IDS design, Cooperating Security Managers [9], uses a network structure, where information flows from any node to any other node, by consolidating the collection, aggregation, and command and control functions into a single component residing on every monitored system. Any significant events occurring at one system that stem from a connection originating from another are reported back to the system manager of the originating system by the security manager at the system where the event occurred. In situations where the originating system of the connection is an intermediate node in a communication chain, the system manager is obliged to report onward to the next system manager in the chain. Because of the potential for unconstrained communication flow, network structures, in general, tend to suffer from communications inefficiency when taken to the extreme (i.e., everyone directly communicating with everyone else). They can however, compensate for this inefficiency with flexibility in function.

1.2 Shortcomings of current intrusion detection systems

Present-day IDSs are less than perfect. Developers continue to address shortcomings through the improvement and refinement of existing techniques, but some shortcomings are inherent in the way IDSs are constructed. The most common shortcomings include the following items:

1.2.1 Lack of Efficiency

IDSs are often required to evaluate events in real time. This requirement is difficult to meet when faced with a very large number of events as is typical in today's networks. Consequently, host-based IDSs often slow down a system and network-based IDSs drop network packets that they do not have time to process.

1.2.2 High Number of False Positives

Most IDSs detect attacks throughout an enterprise by analyzing information from a single host, a single application, or a single network interface, at many locations throughout the network. False alarms are high and attack recognition is not perfect. Lowering thresholds to reduce false alarms raises the number of attacks that get through undetected as false negatives. Improving the ability of an IDS to detect attacks accurately is the primary problem facing IDS manufactures today.

1.2.3 Burdensome Maintenance

The configuration and maintenance of intrusion detection systems often requires special knowledge and substantial effort. For example, misuse detection has usually been implemented using expert system shells that encode and match signatures using rule sets. Upgrading rule sets involves details abnormal to the expert system and its language for expressing rules sets, and may permit only an indirect specification of the sequential interrelationships between events. Similar considerations may apply to the addition of a statistical metric, typically used for detecting unusual deviations in behavior.

1.2.4 Limited Flexibility

Intrusion detection systems have typically been written for a specific environment and have proved difficult to use in other environments that may have similar policies and concerns. The detection mechanism can also be difficult to adapt 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. Often the IDS needs to be completely restarted in order to make changes and additions take effect.

1.2.5 Vulnerability to Direct Attack

Because of the reliance on hierarchical structures for components, many IDSs are susceptible to attack. An attacker can cut off a control branch of the IDS by attacking an internal node or even decapitate the entire IDS by taking out the root command and control node. Typically, such critical components reside on platforms that have been hardened to resist direct attack. Nevertheless, other survivability techniques such as redundancy, mobility, dynamic recovery, etc. are lacking in current implementations.

1.2.6 Vulnerability to Deception

A network-based IDS evaluates network packets using a generic network protocol stack to model the behavior of the protocol stack of the hosts that it is protecting. Attackers take advantage of this inconsistency by sending specially adapted packets to a target host, which are interpreted differently by the IDS and by the target. This can be done in various ways such as altering fragmentation, sequence number, and packet flags. The attacker penetrates the target while the IDS either is blind to the attack or fooled into interpreting that the target resisted the attack.

1.2.7 Limited Response Capability

IDSs have traditionally focused on detecting attacks. While detection serves a useful purpose, oftentimes a system administrator is not able to immediately analyze the reports from an IDS and take appropriate action. This gives an attacker a window of opportunity in which to freely operate before being countered by the actions of the administrator. Many IDSs are beginning to implement automated response capabilities to reduce significantly the time available for attackers to extend their grip on a network. However, they are limited in their ability to adapt dynamically to an attack.

1.2.8 No Generic Building Methodology

In general, the cost of building an IDS from available components is considerable, due in large part to the absence of a structured methodology. No such structuring have emerged from the field itself. This may be partly a result of a lack of common agreement on the techniques for detecting intrusions.

1.3 Agents

We can define an agent as anyone or anything that acts as a representative for another party, for the express purpose of performing specific acts that are seen to be beneficial to the represented party. A software agent, which has been around for approximately twenty five years, is a software program that performs tasks for its user within a computing environment. Technically speaking, most fourth generation software applications could be defined as agents. Everyday we ask computers, through software, to perform hundreds of different tasks for us, essentially calling upon their agency attributes. As we descend deeper into the concept of agency, we can see that there are distinct characteristics that collectively constitute a software agent. Software agents are differentiated by other applications by their added dimensions of mobility, autonomy, and the ability to interact independent of its user's presence.

1.3.1 Software Agents

A software agent is loosely defined as a program that can exercise an individual's or organization's authority, work autonomously toward a goal, meet, and interact with other agents. A software agent comprises the code and state information needed to carry out some computation and requires an agent platform to provide the computational environment in which it operates.

1.3.2 Types of Agents

There are two types of agents.

1. Static agents
2. Mobile agents

As we are using mobile agents so leaving the discussion of static agents, we directly come to mobile agents

1.3.2.1 Mobile Agents

Stationary agents remain resident at a single platform, while mobile agents are capable of suspending processing on one platform and moving to another, where they resume execution of their code. Mobile software agents provide a new and useful paradigm for distributed computing. Unlike the client-server computing paradigm, relationships among entities tend to be more dynamic and peer-to-peer, stressing autonomous collaboration.

A significant number of mobile agent systems have been developed at universities and by industry. Although mobile agents retain the characteristics of autonomy and collaboration as with intelligent agents, emphasis is on mobility characteristics, often relying on simple straightforward algorithms for reasoning and collaboration through less elaborate interpretation of messages.

Mobile agents potentially provide various advantages for intrusion detection and response systems, in particular over more traditional monolithic intrusion detection systems (IDS). Although large part of the disadvantages of traditional IDS can be addressed by distributed IDS systems, what seems promising and deserving of pointing out is the fact that mobile agents may allow to perform some tasks better than with other technologies.

This research is to develop mobile agent (MA) architecture for distributed detection and response to attacks in large-scale networks. It will exploit the flexibility of MA to more efficiently detect and respond to intrusions by significantly reducing the time to do so. Other required MA characteristics, such as dependability and portability are also important for ID but will be addressed as a secondary objective in the research. However, the architecture to be developed is an application architecture and dependent of OS platform (Linux) and MA system environment (D' Agents).

This research is new in the sense that it will fully exploit the mobility of MA in complement to existing IDS systems.

1.4 Mobile Agent Technology

IDSs implemented using MAs is one of the new paradigms for intrusion detection. MAs are a particular type of software agent, having the capability to move from one host to another. A software agent can be defined as

"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 cooperate with them, and perhaps move from place to place in doing so."

Mobile agents have been a research topic of interest for several years, yet this research has for the most part remained within laboratories and has not experienced a wide-scale adoption by industry. The development of the World Wide Web application, however, has dramatically stimulated interest in this area of research by offering the possibility of a widely deployed application that could use mobile agent technology. The research community visualizes mobile agents launched via web browsers to gather information and interact with any node in the network. IBM and General Magic were early pioneers of this vision, [7,8].

An important observation to make about most of the early work in this field is the assumption made by most researchers about a totally open system. That is, the security problems being addressed are those found in a system with open connectivity and with the maximum possible threats. Several researchers reached conclusions indicating that the paradigm was not useful since there were always certain threats that could not be adequately countered while maintaining a totally open system. Partly because of these conclusions, as well as well publicized attacks against early Java-enabled systems, security related problems have hindered the widespread adoption of MA technology. Security architectures have been defined, but they contain too much residual risk for most applications. Recent work at the University of Tulsa, for example, proposes using mobile agents for data mining purposes. Such an application requires providers of information to keep their systems "open" to a multitude of users, most of whom are unknown to the host. A good overview of current mobile agent projects and technology is provided in [9].

However, relatively little work has been done on using a mobile agent architecture for the purpose of providing a security capability, such as intrusion detection. If a mobile agent architecture is designed for a specific purpose such as system administration or security function maintenance, then strong authentication may be enforced and the residual risk decreases significantly.

While MAs are an extraordinarily powerful tool, their implementation has been hindered by security considerations. These security considerations are especially critical for intrusion detection systems, with the result that most security research in this field has concentrated upon the architecture necessary to provide security for mobile agents. We claim that such negative results are not fatal to the proposed study since these security issues are likely to be addressed by the research community and there will be few authorized users of the MA-based IDSs within an organization.

1.4.1 Mobile Agents for Intrusion Detection

For mobile agents to be useful for intrusion detection, it is necessary that many, if not all, hosts and network devices are installed with an MA platform. This is not a far-fetched assumption because an MA platform is general-purpose software that enables organizations to implement many different applications. If MAs become popular, every new host may come preinstalled with a MA platform just as today most personal computers come bundled with a Java interpreter in the web browser. Contrast this to many IDS schemes that assume that a host-based IDS is installed on every host. It is generally too expensive to install a proprietary solution (like a host-based IDS) on every host in a network, but it is not unusual to install a general-purpose interpreter on every host.

1.4.2 Advantages of using Mobile Agents

A number of advantages of using mobile code and mobile agent computing paradigms have been proposed [11,12]. These advantages include: overcoming network latency, reducing network load, executing asynchronously and autonomously, adapting dynamically, operating in heterogeneous environments, and having robust and fault-tolerant behavior. This section examines these claims and evaluates their applicability to the design of ID systems.

1.4.2.1 Overcoming Network Latency

Mobile agents are useful for applications that need to respond in real time to changes in their environment, because they can be dispatched from a central controller to carry out operations directly at the remote point of interest. In addition to detecting and diagnosing potential network intrusions, an IDS needs to provide an appropriate response in order to protect and defend the network from malicious behavior. While a central controller can send messages to the nodes within the network and issue instructions on how to respond to a particular condition or perceived threat, the approach is problematic. For example, the central controller may have to respond to a number of events throughout the network in addition to handling its normal processing load and become a bottleneck or a single point of failure. If connections to this central server are slow or unreliable, the network communications are susceptible to unacceptable delays. Mobile agents, since they are distributed throughout the network, may take advantage of alternate routes around any problem communication links. It will always be faster to send a message to a network node to execute predetermined, resident code, rather than send a mobile agent to the node. However, such an architecture requires that all response and reconfiguration actions be predefined, replicated and distributed throughout the network. The response mechanism then constitutes, in effect, a large distributed database, raising serious administration problems concerning configuration management, consistency, and transaction control. Innovative responses, by definition, must be transmitted at least once to each affected node, either by conventional or network means, a series of messages, or by a mobile agent. Of these choices, the mobile agent technique offers the fastest response.

1.4.2.2 Reducing Network Load

One of the most pressing problems facing current IDSs is the processing of the huge amounts of data generated by the network traffic monitoring tools and host-based audit logs. IDSs typically process most of this data locally. However, abstracted forms of the data are often sent to other network locations where the data is further abstracted and then eventually sent to a central processing site that evaluates abstracted results from all location in the network. Even though the data is usually abstracted before being sent out on the network, the amount of data can still place a considerable communication load on the network. Mobile agents offer an opportunity to reduce the network load by eliminating the need for this data transfer.

Instead of transferring the data across the network, mobile agents can be dispatched to the machine on which the data resides, essentially moving the computation to the data, instead of moving the data to the computation, thus reducing the network load for such a scenario. Clearly, transferring an agent that is smaller in size than the data to be transferred reduces the network load. These benefits hold when the comparison is made between encrypted lightweight mobile agents and the relatively larger data to be transferred.

1.4.2.3 Asynchronous Execution and Autonomy

IDS architectures that are coordinated by a central host require 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. Mobile agent frameworks allow IDSs to continue operation in the event of the failure of a central controller or communication link. Unlike message passing routines or Remote Procedure Call (RPC), once the mobile agent is launched from a home platform it can continue to operate autonomously even if the host platform from where it was launched is no longer available or connected to the network. The coordination of IDS sensors and filters can be protected from the loss of network connections since the mobile agents do not require control by another process. A mobile agent's inability to communicate with central controller would not prevent it from carrying out its assigned tasks.

Although disconnected operation is possible, a number of issues need to be addressed. Distributing the functions of a central controller among the network components is a non-trivial problem. Another problem concerns the operational methods of MAs themselves. For example, Java-based MAs typically load their class files dynamically, as needed, from their home platform. The ability to dynamically load classes also has security.[12] implications. If the home platform is not available, these class files may be provided by the local host or must be found and transferred from a remote trusted host. Class loading from a remote platform or the local host platform raises a number of security issues. The class files may have been modified in such a way as to alter the functionality of the agent or even to allow for eavesdropping of the agents' transactions. Class versioning problems may also yield problems from which the MAs may be able to recover.

1.4.2.4 Structure and Composition

MAs allow for a natural way to structure and design an IDS. For example, rather than a monolithic static system, an IDS can be divided into data producer and data analyzer components and represented as agents. The data producer provides an interface to the networks it sniffs or audit trails it filters. Multiple analyzers, each responsible for detecting a single attack or a small set of attacks, interact with the producer to look for attacks. Under such a framework, MAs from multiple vendors can be used to create an IDS. If a company has the best detector for attack X and another company has the best detector for attack Y, then we can use MAs from both vendors to detect X and Y. Even where manufacturers do not produce agent-based products, it may be possible to reconstitute the product as an agent through wrapping or other techniques. In such an environment, users can also write customized MAs to detect events specific to their environment and work seamlessly with the other MA components. Although this approach applies equally as well to an IDS composed of static components, the agent orientation and mobility considerations provide inherent motivation for identifying and compartmentalizing functionality.

1.4.2.5 Adapting Dynamically

Just as the network's configuration, topology, and traffic characteristics change over time, so should the types of network tests performed and activities monitored. Each computing node in the network will require different tests and these tests will change over time; some tests will no longer be necessary, while new tests will need to be added to the test suite as new vulnerabilities and threats evolve. MAs provide a versatile and adaptive computing paradigm as they can be retracted, dispatched, cloned, or put to sleep as network and host conditions change. For example, as better MAs detectors for an attack are developed they can be sent out on the network to replace the older version, or if an MA is producing too many false positives it can be recalled or gracefully terminated. MAs also have the ability to sense their execution environment and autonomously react to changes. For example, if the computational load of the host platform is too high and the host's performance doesn't meet the agent's service expectations, the agent and its data can move to another machine that can better satisfy its computational needs. MAs can distribute themselves among the hosts in the network in such a way as to maintain the optimal configuration for solving a particular problem.

1.4.2.6 Operating in Heterogeneous Environments

Large enterprise networks are typically comprised of many different computing platforms and computing devices. One of the greatest benefits of MAs is the implementation of interoperability at the application layer. Interoperability at the computer or transport layer, such as solutions provided by single vendors, requires significant changes to the host's environment. Interoperability at the presentation layer, such as the CIDF model [13], limits flexibility in updating the system for new attacks. Conversely, while MA frameworks must be installed on each host, MAs themselves are independently configurable. Since mobile agents are generally computer and transport-layer independent, and dependent only on their execution environment, they offer an attractive approach for heterogeneous system integration. MAs' ability to operate in heterogeneous computing environments is made possible by a virtual machine or interpreter on the host platform. Data fusion efforts can be facilitated by having mobile agents run on switches, routers, and other networking elements. MAs can run on any computing node that can host an agent platform. The ability of MAs to operate in heterogeneous environments also provides an opportunity for the easy integration of network-based and host-based tools operating on various platforms. COTS interoperability may also be facilitated via the use of Agent Communication Languages (ACL) designed for network security testing and intrusion detection domains. Although the MA framework allows an IDS to operate in heterogeneous environments, the tests performed or tasks assigned to the mobile agents are for the most part platform-dependent. Therefore, unless a common programming interface for intrusion detection functions is available, agents must either be restricted to a single class of host or be designed to accommodate heterogeneity in some fashion (e.g., dynamically load or intrinsically convey the host dependent code).

1.4.2.7 Robust and Fault-tolerant Behavior

The ability of mobile agents to react dynamically to unfavorable situations and events makes it easier to build robust distributed systems. For example, if a host is being shut down, all agents executing on that machine are warned, whenever possible, and given time to dispatch and continue their operation while preserving their execution state on another host in the network. Their support for disconnected operation and distributed design paradigms eliminate single point of failure problems and allow mobile agents to offer fault-tolerant characteristics. While there are many features of MAs that enable applications to be robust and fault-tolerant, we should mention a few drawbacks. The ability of the mobile agents to move from one platform to another in a heterogeneous environment has been made possible by the use of virtual machines and interpreters. Virtual machines and interpreters, however, can offer only limited support for preservation and resumption of the execution state in heterogeneous environments because of differing representations in the underlying hardware. For example, the full execution state of an object cannot be retrieved in Java. Information such as the status of the program counter and frame stack is currently forbidden territory for Java programs. Conventional fault-recovery techniques aren't sufficient for the mobile agent computing paradigm. For example, check pointing before and after arrival, and upon completion of certain transactions or events may be necessary to ensure for acceptable fault-recovery. With each check-pointing procedure and non-repudiation mechanism invoked, however, more overhead is introduced. Even though an arsenal of techniques exist to provide

security and fault-tolerance, the designer must be careful in selecting which mechanisms to use and how they impact the overall system performance and functionality.
Although mobile agents possess a great deal of autonomy and perform well in disconnected operations, the failure of the home platform or other platforms that the agents rely on to provide security services can seriously reduce their intended functionality. Even though a mobile agent can become more fault-tolerant by moving to another machine, the mobile agent's reliance on the safe operation of a safe home or trusted platform places restrictions on its functionality. Designers of mobile agent platforms are also faced with tradeoffs between security and fault-tolerance. For example, in order to address the security risks involved in "multi-hop" agent mobility, some agent architectures have been built on centralized client-server models requiring agents to return to a central server before moving on to another host machine. Clearly, addressing the security risks in this manner renders all the mobile agents vulnerable to a failure of the central server.

1.4.2.8 Scalability

The computational load on centralized IDSs increases as more processing nodes are added to the networks they monitor. As networking technology continues to improve, increased bandwidth and network traffic will place greater demands on these centralized architectures. Distributed MA IDS architectures are one of several options that allow computational load and diagnostic responsibilities to be distributed throughout a network. As the number of computing elements in the network increases, agents can be cloned and dispatched to new machines in the network.

1.5 Project Overview

In developing Intrusion detection and response system (IDRS), we propose a new intrusion detection and response model. IDRS will reduce the overhead of the system and detect new or unknown forms of attack. Our goal is not to detect all intrusions precisely but to detect many intrusions efficiently. To accomplish this goal, our system works by watching events that may relate to intrusions named as Marks Left by Suspected Intruder (MLSI). An MLSI is found when one of these events will occur.

1. Modification of critical files such as /etc/passwd, /etc/shadow, /etc/hosts.equiv and /.rhosts
2. su -id command was issued

Instead of analyzing all of the users' activities, if an MLSI is found, IDRS will gather information related to the MLSI, analyze the information, and decide whether an intrusion has occurred. For example, IDRS monitors whether or not critical files related to system security have been modified, since, in many cases, intruders tamper with them. However, because legitimate users may also change the files, the system cannot conclude solely based on file modifications that an intrusion has occurred. IDRS therefore gathers further information related to the modification of the file before deciding if an intrusion has occurred.

1.6 Architecture

In many present day conventional network intrusion detection systems, each target system transfers its system log to an intrusion-detection server, and the server analyzes the entire log in search of intrusions. In a large-scale network deploying an intrusion detection system, network traffic will be extremely high, since the volume of the system logs that are routinely transferred is very large, though most of it has no information related to intrusions. Therefore, this type of intrusion detection system on a large-scale network does not fulfill its function efficiently. To solve this problem, we adopted a mobile-agent paradigm in developing IDRS. Mobile agents autonomously migrate to target systems to collect only information related to intrusions, eliminating the need to transfer system logs to the server. IDRS consists of a manager, monitoring agent, manager logs, information Log, route-tracing agents, and information-gathering agents.

1.6.1 Monitoring Agent

The monitoring agent, present on each target system, monitors system logs in search of MLSIs. If a monitoring agent finds an MLSI, it reports this finding to the manager. The monitoring agent also reports on the type of MLSI.

1.6.2 Route Tracing Agent (RTA)

The intrusion-route tracing agent traces the path of an intrusion and identifies its point of origin, the place from which the user leaving an MLSI remotely logged onto the target host. In the course of finding the origin, a tracing agent can find any intermediate nodes that were compromised. When a RTA goes to the next system, first it checks information Log. If there is no information in the information Log about this particular MLSI, the RTA then completes its task, enters information in the information Log and moves on to the next compromised system. If information already exists in the information Log information related to that MLSI, meaning that another agent has already traced this particular MLSI. Then the tracing agent enters its reference in the information Log and returns to the manager. At a specific point, where RTA cannot proceed to the next system, it means that this particular system is the origin of the intrusion. So RTA returns to the manager after putting information into the information Log.

1.6.3 Information Gathering Agent (IGA)

An information-gathering agent collects information related to MLSIs from a target system. Each time a RTA in tracking down of an intruder is dispatched into a target system by the manager, it activates an information-gathering agent in that system. Then the information-gathering agent collects information depending on the type of MLSI, returns to the manager, and reports.

If the RTA migrates to another target system, it will activate another information-gathering agent on that system, which will gather information on that system. Many information-gathering agents may be activated by many different RTAs on the same target system. An information-gathering agent is not capable of deciding whether an intrusion has occurred or not. An Information gathering agent posts the following information onto the manager log. Depending on that information manager decides whether an intrusion has really occurred or not. IGA puts the following information on the manager log.

1. ID of the information gathering agent.
2. The name of the target system from where information was gathered.
3. The name of the target system preceding the target where information was gathered.

4. Information gathered on the target system

1.6.4 Manager

The manager analyzes information gathered by information-gathering agents and detects intrusions. It manages Manager log and process of dispatching the RTAs and provides an interface between administrators and the system. The manager accumulates and weighs the information entered by the IGAs on the manager log, and if the weights exceed a set threshold, the manager concludes that an intrusion has occurred. After dispatching RTAs, the manager has no concern with the movement of RTAs. i.e. where the RTA will go after a specific system.

1.6.5 Manager log

This is on the manager's machine, which is a means of exchanging information among IGAs because it is shared area, accessible by all IGAs. Information gathering agents (IGAs) put information in an unarranged manner (not with respect to intrusion route) into the manager log independently. This unorganized mass of information is arranged in the manager log.

1.6.6 Information Log

It is also shared area, which exists on every target system, used by RTAs for exchanging information. Any RTA can know whether a track under its inspection has already been traced by other agents, and can use this information in deciding where to go because the manager have no concern with the migration of RTAs. Therefore, many RTAs may trace the same intrusion. To avoid this overlapping, RTAs exchange information with each other regarding their respective information. Tracing agents employ the information Log for exchange of information to avoid re-tracing, RTAs put following information in the information Log.

1. Name of the following system
2. Process ID in the following system
3. Time stamp when the user logged onto the target system

4. Process ID leaving the MLSI, or the name of the target system where the tracing agent begins the trace
5. Tracing agent's ID
6. Time stamp when the agent began the trace

1.7 How it works

The manager, monitoring agent, and RTA work together in the following way.

1. Each monitoring agent on the target system seeks an MLSI from the system log.
2. If the monitoring agent detects an MLSI, it reports to the manager.
3. The manager dispatches a RTA to the target system where the MLSI was detected. The RTA arrives at the target system and activates an information-gathering agent.
4. After activating the information-gathering agent, the tracing agent investigates the point of origin of the MLSI in an effort to identify the user's remote site. The tracing agent can derive this from the accumulated data about network connection and processes running on the system.
5. The information-gathering agent collects information related to the MLSI on the target system.
6. After collecting information, the information-gathering agent, independent of the tracing agent, returns to the manager, and enters the information on the manager log.
7. The RTA moves to the next target system on the tracing route, and it activates a new information-gathering agent.
8. If the tracing agent arrives at the origin of the route, or cannot move anywhere, or if other RTAs have chased the route it could follow, it returns to the manager and puts the report about its bearings on the manager log.

When many MLSIs are found by a single target system in different sessions over a short period, many RTAs corresponding to the MLSIs are launched into the target system. A RTA is not able to make judgments about intrusions, and is not capable of deciding whether an intrusion has occurred.

1.7.1 Overlap Tracing

A tracing agent begins to trace from the point in a target system where an MLSI is first detected. If a user who leaves the MLSI leaves another MLSI on his way to the target, another tracing agent will be dispatched. For example, suppose user X remotely logs onto target systems A, B, C, and D in this order: A - B - C - D. User X compromises the systems and MLSIs are detected on targets D and B respectively.

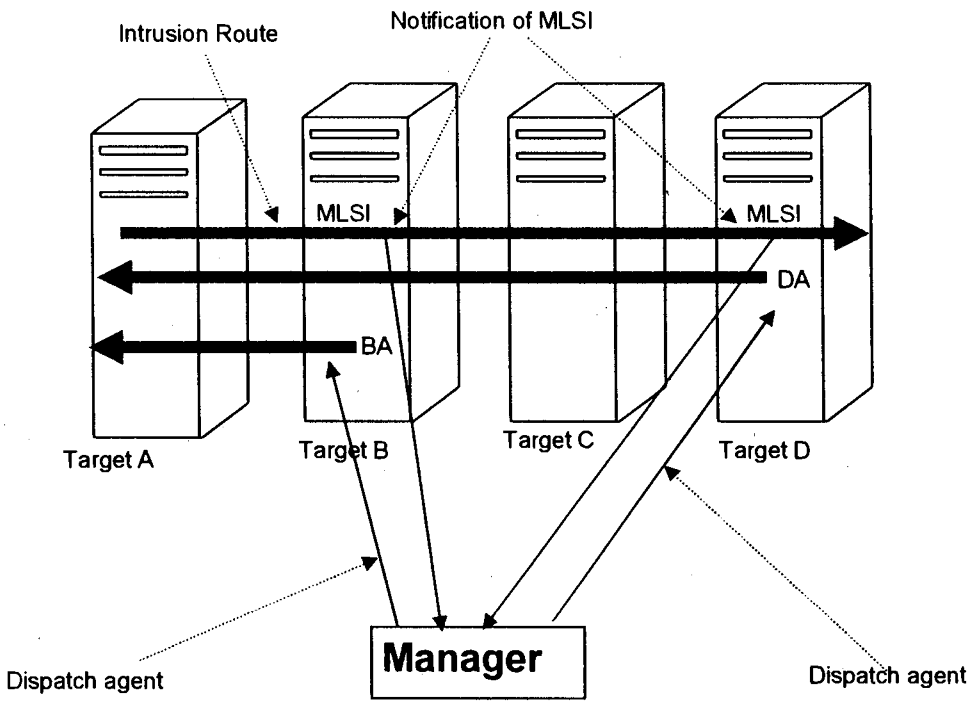


Figure 1.2: Overlap Trace.

The sensors of D and B report to the manager independently, and the manager dispatches tracing agents to both targets D and B. The tracing agent DA traces intrusions in the following order: D - C - B - A. Tracing agent BA, on the other hand, traces in the following order: B - A. The two agents' tracings therefore overlap on B – A

1.7.2 Response Mechanisms

Our main emphasis was to detect intruders but we gave a light touch to response to intruders. However, humans cannot automate what they themselves cannot do. For responding the detected intruders, we present following mechanisms.

1. Response at the host level
2. Response at the network level

1.7.2.1 Response at the host level

Responding at the host level, the following two mechanisms may be applied

1. Responding at the target
2. Responding at the source

1.7.2.2 Responding at the target

After detecting an attack, it is essential to automatically respond at the target host. A quick response can prevent the attacker from establishing a better foothold and using the penetrated host to further compromise the network. It can also minimize the effort needed to recover damage done by the attacker.

1.7.2.3 Responding at the source

Responding at the attacker's host gives IDRS a much greater power to restrict the attacker's actions. Without using mobile agents, it is unlikely that an IDRS would have sufficient access to an attacker's host in order to take corrective action. While this option has limitations, since it requires an agent platform be active on the attacker's host and the attack to come from within the management domain, it also has the potential to be a very effective part of the IDRS.

At network level, there are further three mechanisms

1. Isolating the target
2. Isolating the source
3. Kill TCP connection between source and target.

1.8 Objectives

The proposed research will respond to the increasing need of organizations to effectively protect their large-scale enterprise networks (intranets, extranets) from malicious intrusions. The work will investigate in mobile agent systems for addressing the following issues:

- How an IDS (intrusion detection system) can be integrated with agent technology for augmenting the ID functions in a network.
- How agents' mobile and autonomous characters can be exploited to support the following functions:
  - Real-time detection and assessment function: Monitoring attacks and assessing their significance.
  - Checking the status of network nodes and of the IDSs deployed onto them, verifying integrity and consistency, and reporting to the main monitoring system.
  - Response function: Take appropriate action in terms of focused intelligence gathering or of protective action by dispatching ID agents.
  - Dependability: Determine how resilience to faults and subversion of IDS can be improved by such a mobile agent system.

The first task of the research is to perform a state-of-the-art analysis of the domain. Subsequent tasks to be addressed are:

- Based on established attack/intrusion classifications, identify the intrusion types which can be best addressed by the system and define a realistic intrusion scenario.
- Define the system architecture.
- Develop a functional prototype.
- Investigate the dependability requirements of the system.

Chapter 2

System Analysis

2 System Analysis

At technical level, software engineering begins with a series of modeling tasks that lead to a complete specification of requirements and a comprehensive design representation for the software to be built. The Analysis model, actually a set of models, is the first technical representation of a system. Over the years many methods have been proposed for analysis modeling. However two of them are now dominate the analysis modeling landscape. The first, *structured analysis* is a classical modeling method and the other approach is *object oriented* method. We have used both modeling techniques for the analysis of the software "Intrusion Detection and Response using Mobile Agent Technology".

2.1 Structured Analysis

Structured analysis is a model building activity. Using a notation that satisfies the operational analysis principles, we create models that depict information (data and control) contents and flow, we partition the system functionally and behaviorally, and we depict the essence of what must be built. Structured analysis was not introduced with a single landmark paper or book that was a definitive treatment of the subject. Early work in analysis modeling was begun in late 1960s and early 1970s, but the first appearance of the structured analysis approach was as an adjunct to another important topic—Structured Design.

2.2 Analysis Model

The Analysis Model must achieve three primary objectives.

1. Describe what is actually required.
2. Establish a basis for the creation of a software design.
3. Define a set of requirements that can be validated once the software is built.

To accomplish these objectives, the analysis model derived during the structured analysis takes the form illustrated in Figure 2.1.

At the core of the model lies the data dictionary — a repository that contains description of all data objects consumed or produced by the software. Three different diagrams surround the core. The entity-relationship diagram (ERD) depicts relationships between data objects. The ERD is the notation that is used to conduct the data modeling activity. The attributes of each data object noted in the ERD can be described using a data object description.

The Data flow Diagram (DFD) serves two purposes:

1. Provide an indication of how data are transformed as they move through the system.
2. Depict the functions and sub functions that transform the data flow.

The DFD provides additional information that is used during the analysis of the information domain and serves as a basis for the modeling of function. A description of each function presented in the DFD is contained in a process specification (PSPEC).

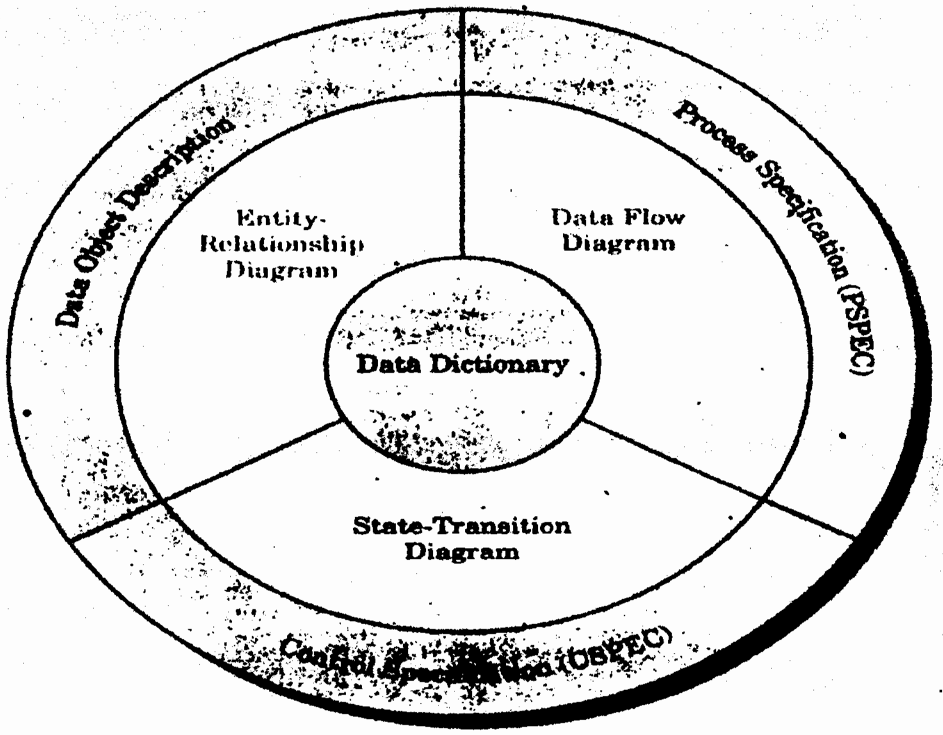

Data Object Description
Process Specification (PSPEC)
Entity-Relationship Diagram
Data Flow Diagram
Data Dictionary
State-Transition Diagram
Control Specification (CSPEC)

Figure 2.1 Analysis Model

T-924

The State-transition diagram (STD) indicates how the system behaves as a consequence of external events. To accomplish this, the STD represents the various modes of behavioral modeling. Additional information about control aspects of the software is contained in the control specification (CSPEC).

2.2.1 Entity Relation Diagram

ERD is used to define the relationship between different entities or objects. These objects are joined with the other based on the relationship they have. ERD focuses solely on data (and therefore satisfies the first operational analysis principle), representing a "data network" that exist for a given system. ERD is especially useful for applications in which data and the relationships that govern data are complex.

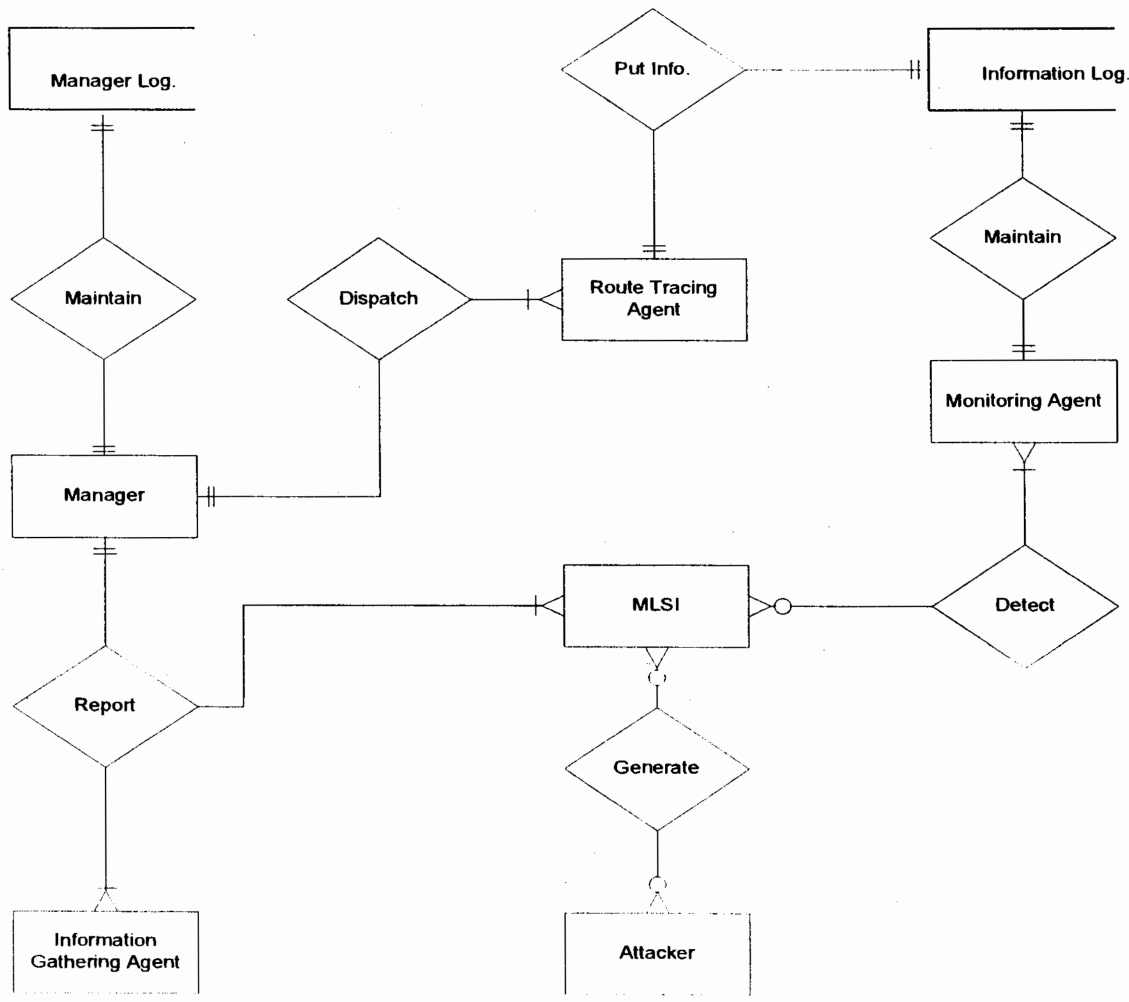


Figure 2.2 ERD for Intrusion Detection and Response using Mobile Agent Technology

2.2.2 Data Flow Diagram

As information moves through the software, it is modified by a series of transformations. A DFD is a graphical technique that depicts information flow and the transforms that are applied as data move from input to output. The DFD is also known as *Data flow graph* or a *bubble chart*.

The DFD may be used to represent a system or software at any level of abstraction. In fact, DFDs may be partitioned into levels that represent increasing information flow and functional detail. Therefore the DFD provides a mechanism for functional modeling as well as information flow modeling. In doing so, it satisfies the second operational analysis principle (i.e. creating a functional model).

A Context Level DFD or Level 0 DFD is called fundamental system model or a context model represents the entire software element as a single bubble with input and output data indicated by incoming and outgoing arrows, respectively. Additional processes and information flow paths are represented as the Level 0. The bubble at level 0 is expanded at Level 1 to reveal the processes and functions in detail and so on. And the information about the functions and processes are provided in PSPEC in either simple English or in PDL form.

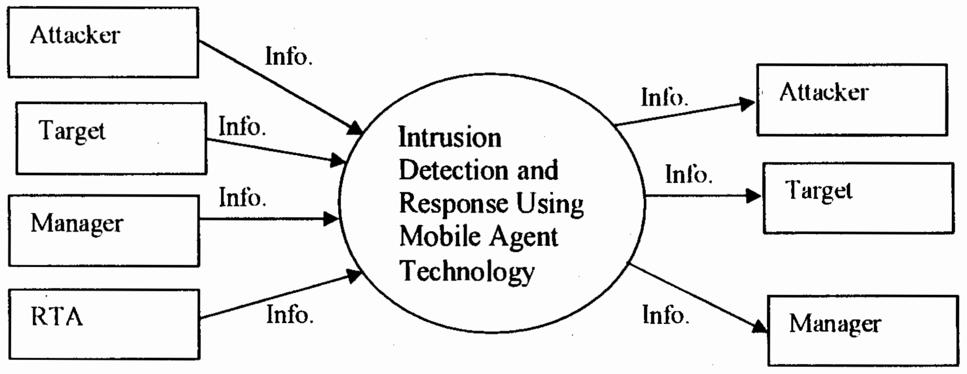


Figure 2.3 Context Level (Level 0)DFD for Intrusion Detection and Response Using Mobile Agent Technology

Figure 2.3 shows the Context Level DFD for the software Intrusion Detection and Response Using Mobile Agent Technology. This level is the highest level of abstraction where no details are shown; only the input to the software and output from software is shown. There is only one bubble which is the software and reveals no function of the software

Now the DFD is expanded and level 1 shows the detail of the process or functions of the software.

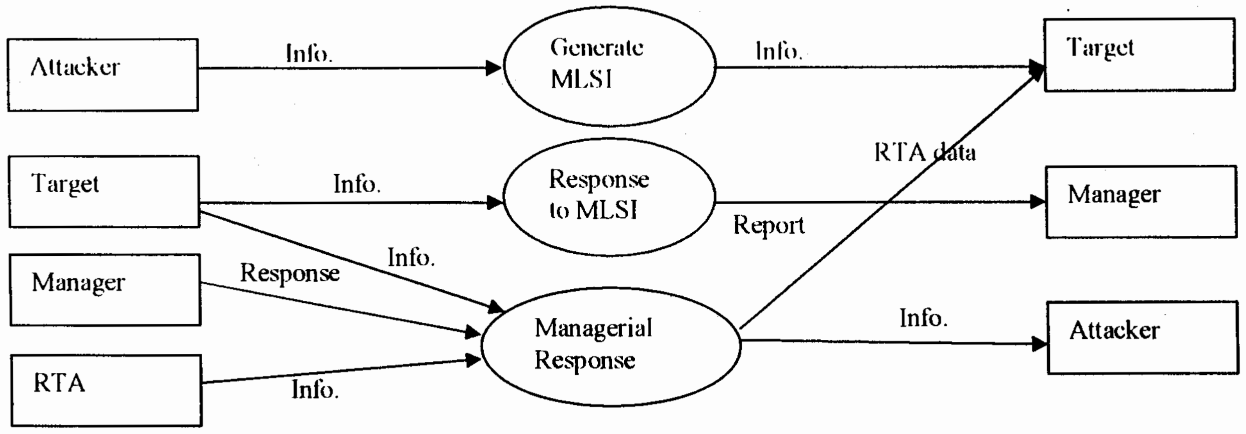


Figure 2.4 Level 1 DFD for Intrusion Detection and Response using Mobile Agent Technology

Figure 2.4 expands the bubble in Level 0 DFD and here the level of abstraction decreases but only up to the functions still the sub functions are not reveled. It also shows the information flow and the arrows, to show which process sends the information and which process use the stored information.

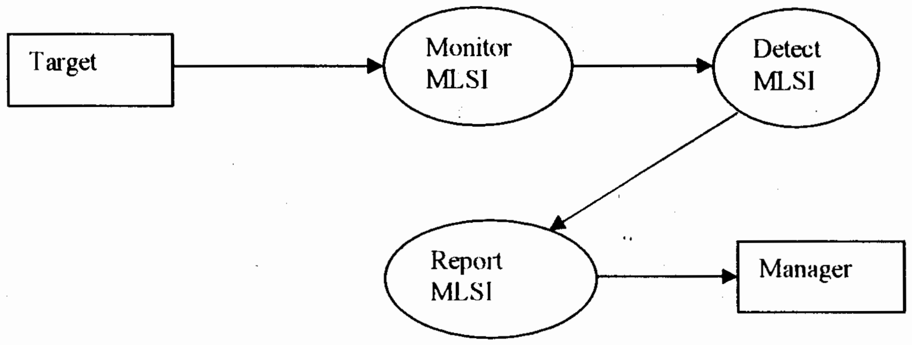


Figure 2.5 Level 2 DFD for response to MLSI

Level 2 DFD process Response to MLSI is shown in Figure2.5. This level shows the sub functions of the process response to MLSI. It shows that the Monitoring Agent will first keep on monitoring all MLSIs and when an MLSI is found it will detect it and then monitoring agent will report the MLSI to the manager.

Now the level 2 DFD for the process managerial response.

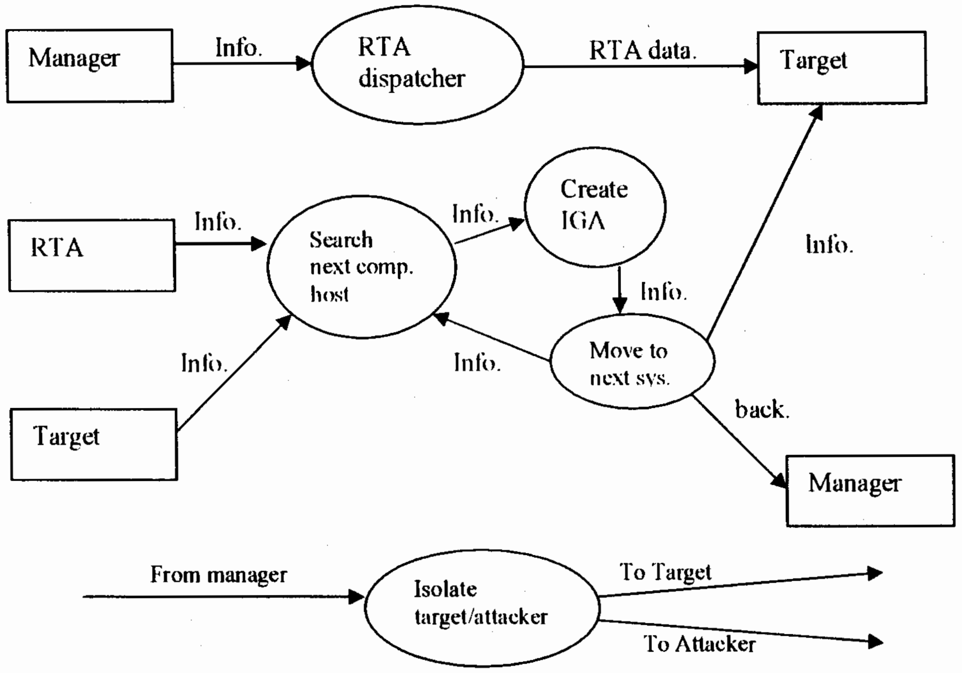


Figure 2.6Level 2 DFD for Managerial Response

Level 2 DFD for the process managerial response is shown in figure 2.6. which shows that how the manager of our system responds when an MLSI is reported to the manager by a monitoring agent. And this level 2 DFD describes the minimum level abstraction and shows the actual functionality. But the process isolate target/source needs more expansion which is further expanded to level 3 in figure 2.7.

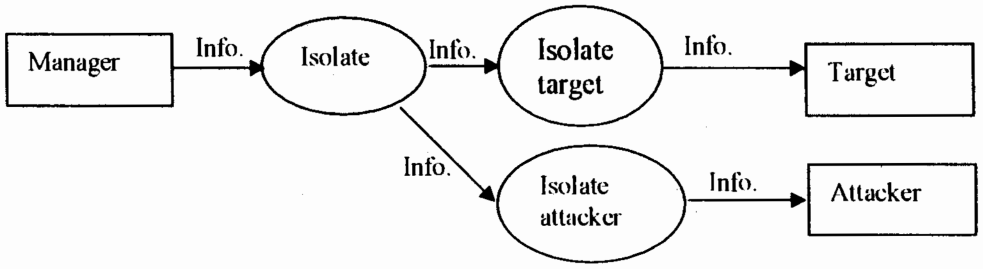


Figure 2.7Level 3 DFD for isolate target/ attacker

Level 3 DFD for the process isolate source/target is shown in figure 2.7.. This shows how the manager will response to the attack. It will either isolate the target or isolate the source. So that the damage may be minimized

More information about the process functionalities is described in PSPEC.

2.2.3 PSPEC

The Process specification contains the detailed information about the processes defined in the DFD. These details either can be in simple English or in Program Design Language (PDL) format. In simple English, the process is defined in simple words while in PDL, the process is written in the format similar to the algorithms but they are not complex as algorithms are. We will define the process in PDL.

2.2.3.1 Initialize the Server

```
Procedure Initialize the server;
    Read the command;
    If command is valid then initialize the server;
    Else do not initialize the server;
    End if;
Endproc
```

2.2.3.2 Interact with user

```
Procedure Interact with user;
    Read the input data;
    If data is valid then send user command to the server;
    Else do not send user command to server;
    End if;
Endproc
```

2.2.3.3 Detect MLSIs

```
Procedure monitor for MLSIs
    Monitor for MLSIs
    If MLSI found
    then send message to manager
        MLSI detected
        Send type of MLSI
        End if;
Endproc
```

2.2.3.4 Dispatch Route Tracing Agent

```
Procedure dispatch RTA
        Send RTA to target host
Endproc
```

2.2.3.5 Detect Route

```
Procedure detect route
        See information log on that host
        If information already exists in information log
        then put ID of RTA in it
               return to the manager
        else search entire log
             find the IP address of the next compromised host
             start an IGA on this machine
             put the info about next compromised host in the information table
             go to next compromised host
        end if;
        if IP address of next compromised host not found
        then return to the manager
        else repeat the procedure
        end if;
endproc;
```

2.2.3.6 Gather Information

```
Procedure gather information
        Get name of target machine
        Get name of next compromised machine
        Get all other information related to MLSI
        Return to the manager
        Put all its belongings onto the manager board
        Put own ID onto the manager board
    Endproc;
```

2.2.3.7 Decide about the MLSI

```
    Procedure decide
        Accumulate the information which is on the manager log
        Check it against set threshold
        If weight exceeds the set threshold
        Then it is Intrusion
            Invoke response procedure
        End if;
    Endproc;
```

2.2.3.8 Response to the intruder

```
    Procedure response
        Isolate the target or isolate the attacker
    Endproc;
```

2.2.4 State Transition Diagram (STD)

The State Transition Diagram indicates how the system behaves as consequence of external events. The labeled transition arrows indicate how the system reacts to the external events as it traverses the defined states. By studying STD, a software engineer can determine the behavior of the system and can ascertain whether there are "holes" in the specified behavior.

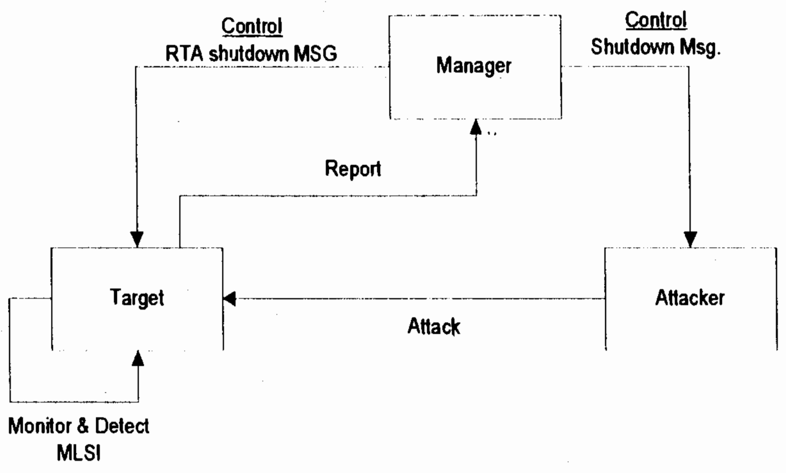


Figure 2.8 State Transition Diagram for the system

2.2.5 Control Specification (CSPEC)

The Control Specification (CSPEC) represents the behavior of the system in two different ways. One is called specification behavior and the second one is called combinational specification. Control Specification contains STD (Specification behavior) and a PAT (Combinational Behavior) table based on the Level 1 DFD. The STD reveals enough information so we do not need to create the PAT. The CSPEC does not give us any information about the inner working of the processes that are activated as a result of this behavior.

2.3 Object-Oriented Analysis

It is a method of analysis that examines the requirements of end-user from the perspective of objects and classes found in the vocabulary of problem domain.

2.4 A Unified Approach to Object-Oriented Analysis

Over the past decade, Grady Booch, James Rumbaugh, and Ivar Jacobson have collaborated to combine the best features of their individual object-oriented analysis and design methods into a unified method. The result, called the *Unified Modeling Language* (UML), has become widely used throughout the industry.

UML allows a software engineer to express an analysis model using a modeling notation that is governed by a set of syntactic, semantic, and pragmatic rules.

In UML, a system is represented using five different "views" that describe the system from distinctly different perspectives. Each view is defined by a set of diagrams. The following views are presented in UML:

- **User Model View.** This view represents the system (product) from the user's (called *actors* in UML) perspective. The use-case is the modeling approach of choice for the user model view. This important analysis representation describes a usage scenario from the end-user's perspective.
- **Structural Model View.** Data and functionality are viewed from inside the system. That is, static structure (classes, objects, and relationships) is modeled.
- **Behavioral Model View.** This part of the analysis model represents the dynamic or behavioral aspects of the system. It also depicts the interactions or collaborations between various structural elements described in the user model and structural model views.
- **Implementation Model View.** The structural and behavioral aspects of the system are represented as they are to be built.
- **Environment Model View.** The structural and behavior aspects of the environment in which the system is to be implemented are represented

In general, UML analysis modeling focuses on the user model and structural model views of the system. UML design modeling addresses the behavioral model, implementation model, and environmental model views.

2.5 Domain Analysis

Analysis for object-oriented systems can occur at many different levels of abstractions. At the business or enterprise level, the techniques associated with OOA can be coupled with a business process engineering approach in an effort to define classes, objects, relationships, and behaviors that model the entire business. At the business area level, an object model that describes the workings of a particular business area (or a category of products or systems) can be defined. At an application level, the object model focuses on specific customer requirements as those requirements affect an application to be built.

We will conduct OOA at a middle level of abstraction. This activity, called *domain analysis*, is performed when an organization wants to create a library of reusable classes (components) that will be broadly applicable to an entire category of applications.

2.5.1 Reuse and Domain Analysis

Object-technologies are leveraged through reuse. The benefits derived from reuse are consistency and familiarity. Patterns within the software will become more consistent, leading to better maintainability. Be certain to establish a set of reuse "design rules" so that these benefits are achieved.

2.5.2 The Domain Analysis Process

Software domain analysis is the identification, analysis and specification of common requirements form a specific application domain, typically for reuse on multiple projects within that application domain ... [Object-oriented domain analysis is] the identification, analysis, and specification of common, reusable capabilities within a specific application domain, in terms of common objects, classes, subassemblies, and frameworks ...

The goal of domain analysis is straightforward: to find or create those classes that are broadly applicable, so that they may be reused.

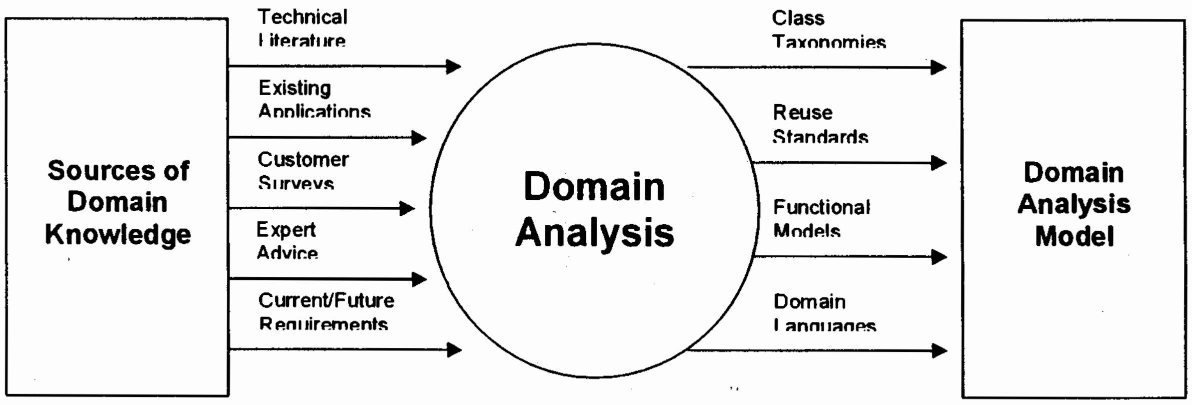


Figure 2.9 Input and Output for Domain Analysis.

Domain analysis may be viewed as an umbrella activity for the software process. By this we mean that domain analysis is an ongoing software engineering activity that is not connected to any one software project. In a way, the role of a domain analyst is similar to the role of master toolsmith in a heavy manufacturing environment. The job of the toolsmith is to design and build tools that may be used by many people doing similar but not necessarily the same jobs. The role of the domain analyst is to design and build reusable components that may be used by many people. Working on similar but not necessarily the same applications.

Figure 2.9 illustrates key inputs and outputs for the domain analysis process. Sources of domain knowledge are surveyed in an attempt to identify objects that can be reused across the domain. In essence domain analysis is quite similar to knowledge engineering. The knowledge engineer investigates a specific area of interest in an attempt to extract key facts that may be of use in creating an expert system of artificial neural network. During domain analysis, *object* (and class) *extraction* occurs.

We performed the following series of activities during the domain analysis process:

2.5.2.1 The Domain to be Investigated

To accomplish this we isolated the business area, system type, or product category of interest. Next, both OO and non-OO "items" were extracted. OO items include specifications, designs, and code for existing OO application classes; support classes (e.g. GUI classes); commercial off-the-shelf (COTS) component libraries that are relevant to the domain; and test cases. Non-OO items encompass policies, procedures, plans, standards, and guidelines; parts of existing non-OO applications; and COTS non-OO software.

2.5.2.2 Categorization of Items Extracted from the Domain

The items were organized into categories and the general defining characteristics of the category were defined. A classification scheme for the categories was proposed and naming conventions for each item were defined. Classification hierarchies were established.

2.5.2.3 Collection of Representative Sample of Application in the Domain

To accomplish this activity, we ensured that the application in question had items that fit into the categories that have already been defined. During the early stages of use of object-technologies, a software organization has few if any OO applications. Therefore, we identified the conceptual (as opposed to physical) objects in each [non-OO] application.

2.5.2.4 Development of Analysis Model for the Objects

The analysis model served as the basis for design and construction of the domain objects.

2.6 The Object-Oriented Analysis Process

The OOA process doesn't begin with a concern for objects. Rather, it begins with an understanding of the manner in which the system will be used—by people, if the system is human-interactive; by machines, if the system is involved in process control; or by other programs, if the system coordinates and controls applications. Once the scenario of usage has been defined, the modeling of the software begins.

A series of techniques is used to gather basic customer requirements and then define an analysis model for an object-oriented system.

2.6.1 Use-Cases

Use-cases model the system form the end-user's point of view. Created during requirements elicitation, use-cases should achieve the following objectives:

- To define the functional and operational requirements of the system (product) by defining a scenario of usage that is agreed upon by the end-user and the software engineering team.
- To provide a clear and unambiguous description of how the end-user and the system interact with one another.
- To provide a basis for validation testing.

2.6.1.1 Use-Cases in the System

A *use-case* is a high level piece of functionality that the system provides. The following are the use-cases in our system.

- Generate MLSI
- Monitor MLSI

- Detect MLSI
- Send Report to Manager
- Dispatch RTA
- Search Next Compromised Host
- Fill Information Log
- Create IGA
- Move to Next System
- IGA report to Manager
- RTA report to Manager
- Is It Intrusion
- Isolate Target
- Isolate Source

2.6.1.2 Actors in the System

An *actor* is anyone or anything that interacts with the system being built. These are the actors in our system.

- Attacker/User
- Target
- Manager

2.6.1.3 Use cases in Expanded Format

- **Use-Case: Generate MLSI**

  Actors: Attacker/User

  Subject: Generate an MLSI

  Summary: User will click on the Generate MLSI button and attacker will perform a malicious action called an MLSI.

  Type: Essential, primary

Typical Course of Actions:

1. This use-case will begin when user will access the machine in the network for which it has no proper privileges.
2. User click the button labeled " Generate MLSI" on attacker machine.

- **Use-Case: Monitor MLSI**

  Actors: Target

  Subject: Monitor for MLSIs on the host

  Summary: This use-case will monitor all malicious activities of remote/local users which has totally no privileges or have no proper privileges to access that particular area of the system to which they are accessing or exceeding their privileges or rights

  Type: Essential, primary

  Typical Course of Actions:

  1. Monitor the activities of users.
  2. When a malicious action is performed by any user ,it will do an action.

  Alternative Courses of Actions:

  1a . If it is normal activity, ignore it, and keep on monitoring

- **Use-Case: Detect MLSI**

  Actors: None

  Subject: Detect an MLSI

  Summary: Detect malicious actions of user which are using the network. And are accessing the target machine without proper privileges.

  Type: Essential , Primary

  Typical Course of Actions:

  1. This Use-Case begins when the Monitor MLSI use-case uses this use-case.
  2. It responds by sending a message to the manager machine.

- **Use-Case: Send Report to Manager**

  Actors: Target

  Subject: Send the report to manager

  Summary: Sends report to the manager about the malicious activity of the user which accessed the system from any node in the network and performed any malicious activity, and penetrated into that area of the target for which he was not allowed to penetrate.

  Type: Primary, Essential

  Typical Course of Actions:

  1. This Use-Case begins when the Detect MLSI use-case uses this use-case.
  2. It responds in a way that it sends a report to the manger machine that a malicious action is performed.

- **Use-Case: Dispatch RTA**

  Actors: Manager

  Subject: Dispatch Route Tracing Agent to Target machine

  Summary: Manager will dispatch a Route Tracing Agent(RTA) to the target machine where an MLSI was detected and was reported by the monitoring agent to the manager.

  Type: Primary , Essential

  Typical Course of Actions:

  1. This use-case begins when the Report to manager use-case reports to the manager machine.
  2. Go to the target machine.
  3. See the entries in the information log.
  4. If information log is empty then start searching the area which was under attack.

- **Use-Case: Search Next Compromised Host**

  Actors: None

  Subject: Search for Next Compromised Host

Summary: This use case will search the next compromised host which was compromised by the attacker before accessing the target so that the system can know which system in the network is source of the malicious action. Because the attacker never wants to leave its identification. It first compromises the node in the network which has less security, then by using that system it compromises the node which has higher security than that and finally using that compromised host it accesses its desired host and accesses the area for which it is not allowed.

Type: Essential, Primary

Typical Course of Actions:

1. This use-case begins when the Dispatch RTA use-case uses this use-case
2. Search for next compromised host.

- **Use-Case: Fill Information Table**

Actors: None

Subject: Fill Information Log

Summary: This use-case fill information Log which is on target machine. Information Log is a shared area on every host in the network.

Type: Primary ,Essential

Typical Course of Actions:

1. This use-case begins when the "Search next compromised host" use-case completes its work .
2. Gather information about next compromised host.
3. Put the information into the information Log.

- **Use-Case: Create IGA**

Actors: None

Subject: Create Information Gathering Agent

Summary: This use-case will gather information about the MLSI. And put that information in a shared area on the manager machine. That is, after completing its task it will go the manager machine and put the information in the manager Log.

Depending on the information, Manager will decide whether an intrusion has actually occurred or it is just a false alarm.

Type: Primary, Essential

Typical Course of Actions:

1. This use-case starts its work when RTA will fill the information Log.
2. RTA will create information gathering agent.
3. Gather information related to MLSI.

- **Use-Case: Move to Next System**

  Actors: None

  Subject: Move Route Tracing Agent to the next compromised host

  Summary: Move Route Tracing Agent to the next compromised host when the task of the route tracing agent has completed.

  Type: Primary, Essential

  Typical Course of Actions:

  1. This use-case will begin when RTA will complete its work.
  2. Move to the next compromised host.

- **Use-Case: IGA Report to Manager**

  Actors: None

  Subject: IGA Report to Manager

  Summary: After gathering information from the target machine about the MLSI, IGA will go back to the manager machine and will put information in the manager Log.

  Type: Primary, Essential

  Typical Course of Actions:

  1. This use case will begin when Information Gathering Agent completes its task on the target machine.
  2. Gather information about MLSI on the target system.
  3. move to the manager machine.
  4. Put the information gathered onto the manager Log.

- **Use-Case: RTA Report to Manager**

  Actors: None

  Subject: RTA report to the manager machine.

  Summary: After Roaming to all the compromised hosts, Route Tracing Agent will reach to such a machine from where it will not find any information about the next compromised host. From this system RTA will go back to the manager machine. And will report about the attacker system.

  Type: Primary, Essential

  Typical Course of Actions:

  1. This use case will start when RTA will complete its task and is on such a system that if finds no compromised host.
  2. Go back to the manager machine.
  3. Put the information about the last system(the system from where RTA came back)

- **Use-Case: Is It Intrusion**

  Actors: Manager

  Subject: Decide MLSI whether it is intrusion

  Summary: When all RTAs have come back to the manager machine dispatched from manager, and all the IGAs which were created by the RTAs. And they put the information on the manager board then manager will decide whether the MLSI was an intrusion or an event of normal access(false alarm).

  Type: Primary, Essential

  Typical Course of Actions:

  1. This use case will be activated when all the RTAs and IGAs came back to manager machine.
  2. Decide whether the MLSI occurred is an intrusion.

Alternative Courses of Actions:

2a. If it is not an intrusion then exit.

- **Use-Case: Isolate Target**

  Actors: Manager

  Subject: Isolate Target machine

  Summary: Cut the TCP/IP connection of the target so that all the network packets delivered for this particular machine may not come to the machine so that damage may be minimized.

  Type: Primary, Essential

  Typical Course of Actions:

  1. This use case will be activated when the manager will decide that the MLSI occurred is an intrusion.
  2. Cut the TCP/IP connection of target.

- **Use-Case: Isolate Source**

  Actors: Manager

  Subject: : Isolate Attacker machine

  Summary: Cut the TCP/IP connection of the Source so that it may not be able to deliver further packets on the network so that damage may be minimized.

  Type: Primary, Essential

  Typical Course of Actions:

  1. This use case will be activated when the manager will decide that the MLSI occurred is an intrusion.
  2. Cut the TCP/IP connection of Attacker

2.6.1.4 Use Case Diagram

Use-Case diagram in Figure 2.10 shows some of the use-cases in the system, some of the actors in the system, and the relationships between them.

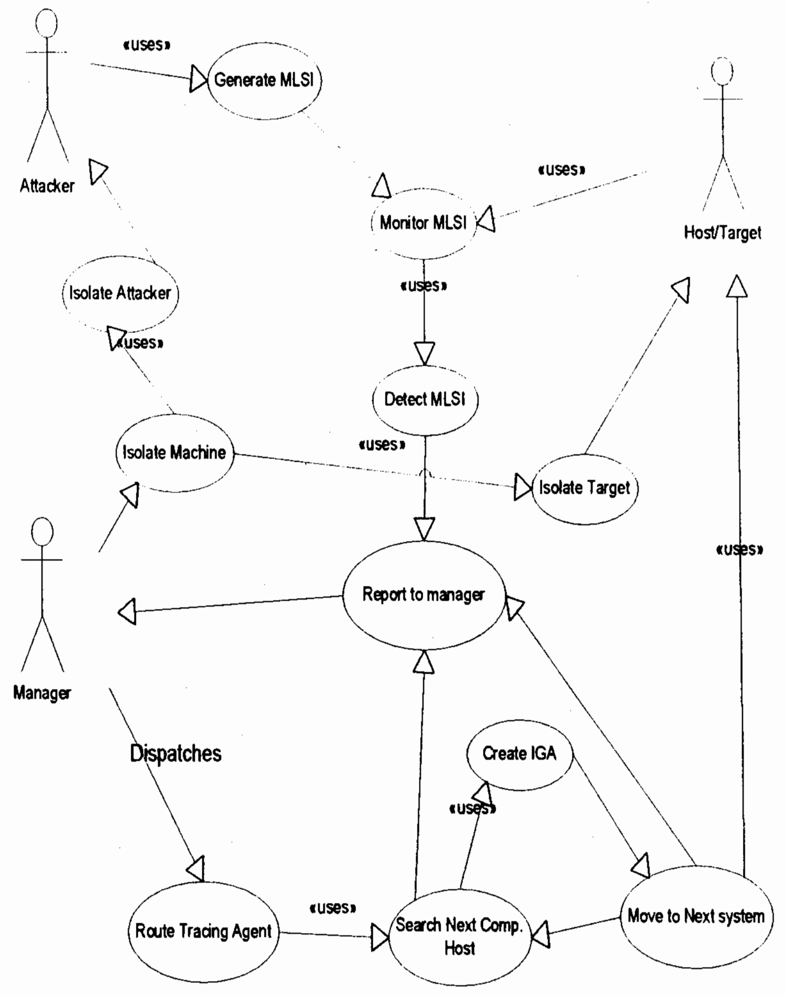


Figure 2.10 Use Case Diagram for Intrusion Detection and Response using Mobile Agent Technology

2.6.2 Conceptual Model

Conceptual Model in Figure 2.11 depicts the concepts found in the domain of the system. In conceptual model we identify the conceptual (as opposed to physical) objects in the application.

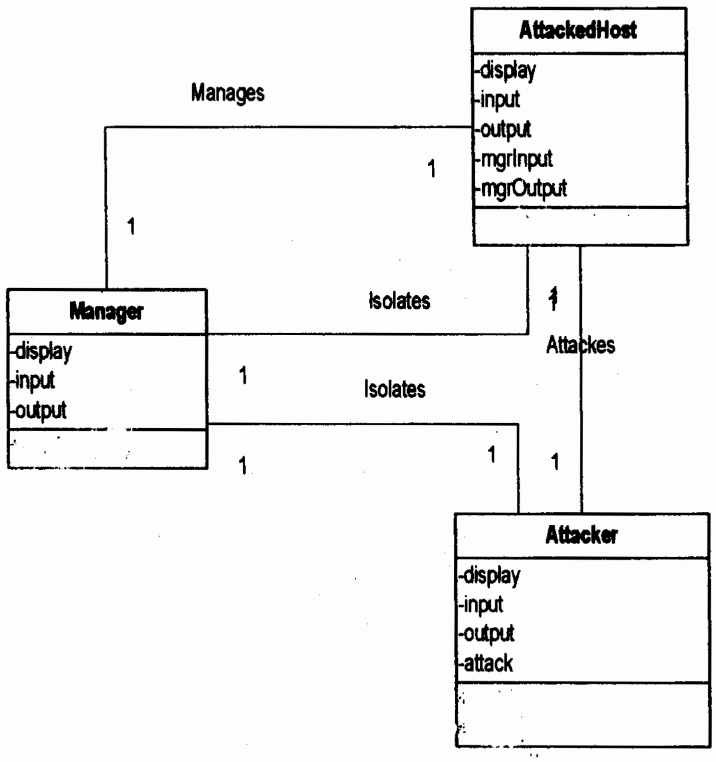


Figure 2.11 Conceptual Model for Intrusion Detection and Response using Mobile Agent Technology

Chapter 3

System Design

3 Design

Design is an iterative process transforming requirements into a "blueprint" for constructing the software. It is the first step in the development phase for any engineered product or system. It can also be defined as "the process of applying various techniques and principles for the purpose of defining a device, a process or a a system in sufficient detail to permit its physical realization."

The designer's goal is to produce a model or representation of an entity that will later be built. The process by which the model is developed combines intuition and judgment based on experience in building similar entities, a set of principles and/or heuristics that guide the way in which the model evolves, a ultimately leads to a final design representation.

3.1 Relation of analysis to design

For design, we need analysis results which serves as base information for design. Infact we explore the analysis in detail and produce design such that it is directly mapped into coding.

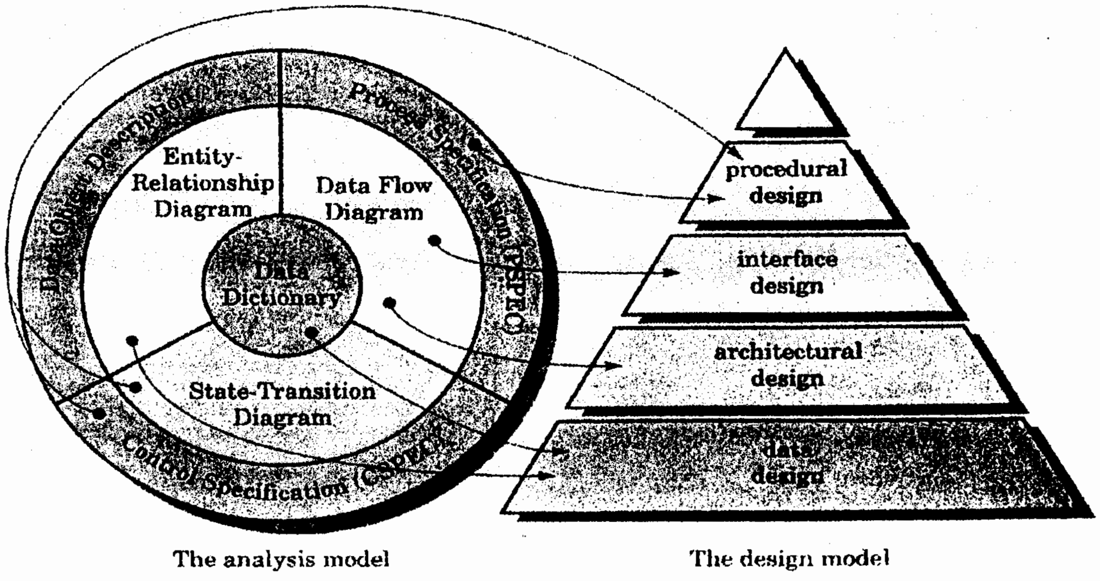


Figure 3.1: Relation of Analysis Model to Design Model.

Figure 3.1 shows the realtion of Analysis model to Design model and the arrows shows which of the information from the analysis model is necessary for which design. Data Design is created using the DataDictionary and Entity-Relationship Diagram information of Analysis model. Archicectural Design is creared using the information from Data Flow Diagram of the Analysis model. Interface Design is also created using the infromation from data flow diagram. Procedural Design uses the information from CSPEC, PSPEC and state-transition diagram of the Analysis model.

3.2 Design Principles

Software design is both a process and a model. The design process is a set of iterative steps that enable the designer to describe all the aspects of the software to be built. It is important to note, however, that the design process is not simply a cookbook. Creative skill, past exprerience, a sense of what makes good software, and an overall commitment to quality are critical success factors for a competent design.

Basic design principles enable the software engineer to navigate the design process. The design principles are:

- The design process should not suffer from "tunnel vision". A good designer should consider alternative approaches, judging each based on the requirements of the problem.
- The design shoud be receable to the anlysis model.
- The design should not reinvent the wheel.
- The design should minimize the intellcetual distance.
- The design should exhibit uniformity and integration.
- The design should be structured to accommodate change.
- The design should be structured to degrade gently, even when aberrant data, events, or operating coditions are encountered.
- Design is not coding, coding is not design.

- The design should be assessed for quality as it is being created, not after the fact.
- The design should reviewed to minimize coneptual errors.

3.3 Design Types

There are four types of Designs.

1. Data Design
2. Architectural Design
3. Interface Design
4. Procedural Design

3.3.1 Data Design

The Data Design transform the information domain model created during analysis into the data structures that will be required to implement the software. The data objects and relationships defined in the entity-relationship diagram and the detailed data content depicted in the data dictionary provide the basis for the data design.

Data Design is the first of four design activities that are conducted during software engineering. The primary activity during data design is to select logical representation of data objects identified during the requirement definition and specification phase. The selection process may involve algorithmic analysis of alternative structures in order to determine the most efficient design or may simply involve the use of a set module that provide the desired operations upon some representation of an object.

3.3.2 Architectural Design

The Architectural Design defines the relationship among major structural elements of the program. This design representation—the modular framework of a computer program—can be derived from the analysis model(s) and the interaction of subsystem defined within the analysis model.

3.3.3 Architectural Design Process

Data flow-oriented design is an architectural design method that allows a convenient transition from the analysis model to a design description of program structure. The transition from information flow to structure is accomplished as part of a five step process:

1. The type of information flow is established.
2. Flow boundaries are indicated.
3. The DFD is mapped into program structure.
4. Control hierarchy is defined by factoring.
5. The resultant structure is refined using design measures and heuristics.

3.3.4 Architecture Design of Software

The Program Structure of Intrusion Detection and Response Using Mobile Agent Technology is show in Figures 3.2, 3.3,3.4, 3.5 of DFD level 1 and 2.

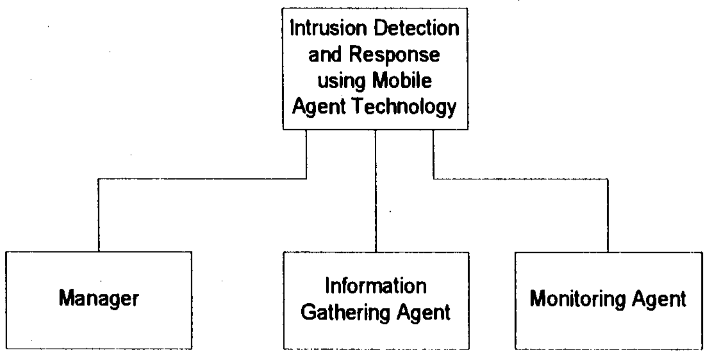

Intrusion Detection and Response using Mobile Agent Technology
Manager
Information Gathering Agent
Monitoring Agent

Figure 3.2 Program structure of intrusion detection and response using mobile agent technology

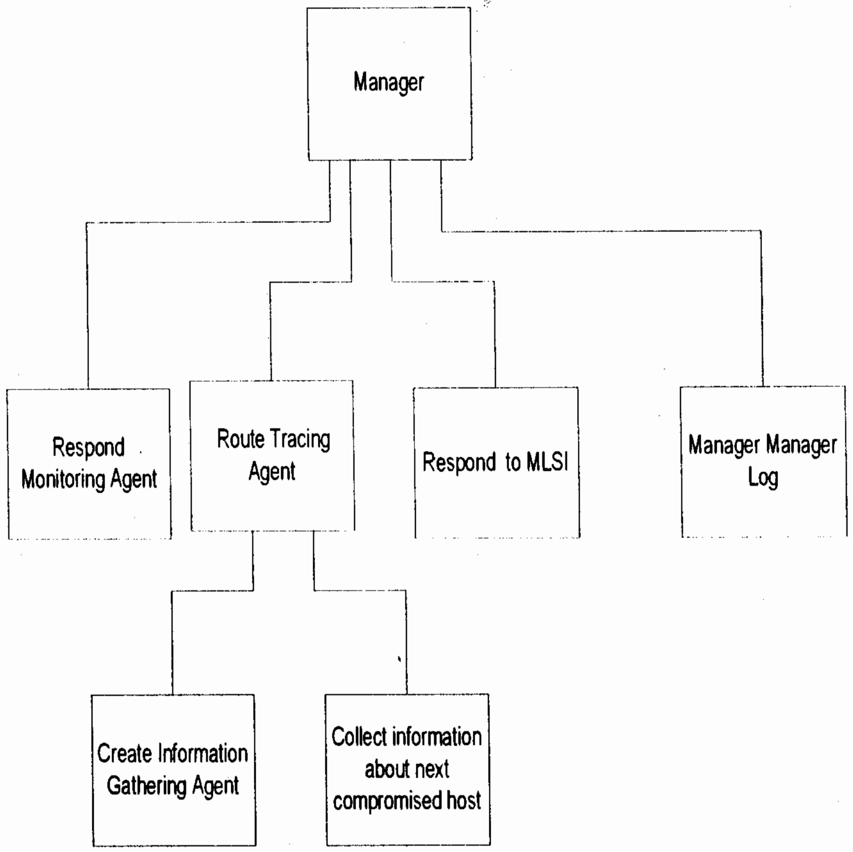


Figure 3.3 program structure of manager

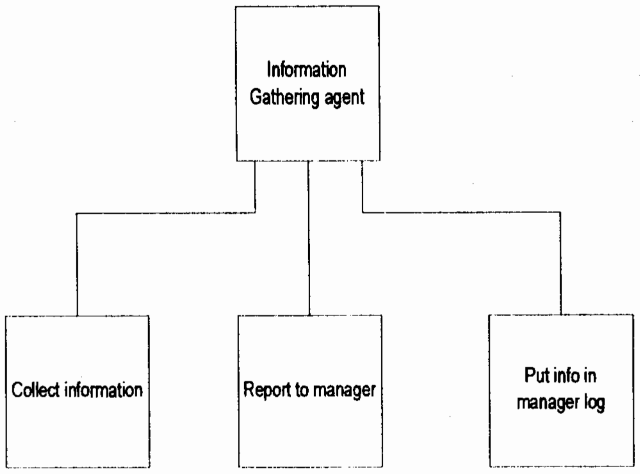


Figure 3.4 Program structure of Information Gathering Agent

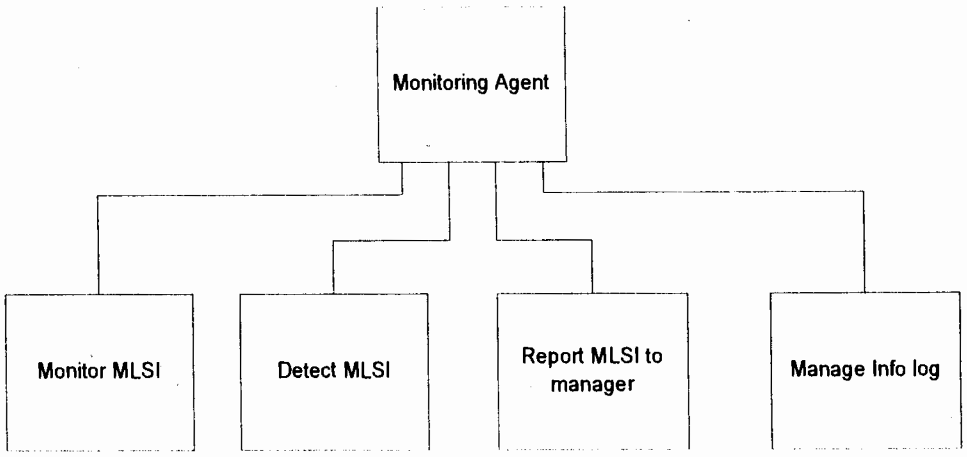


Figure 3.5program structure of Monitoring Agent

3.3.5 Interface Design

The interface design describes how the software communicates within itself, to systems that interoperate with it, and with humans who use it. An interface implies a flow of information (e.g. data and/or control). Therefore, the data and control flow diagrams provide the information required for interface design. But as far as the user-interface is concerned, our software is a service and services don't have user-interface.

3.3.6 Procedural Design

The procedural design transforms structural elements of the program architecture into a procedural description of software components. Information obtained from the PSPEC, CSPEC and STD serve as the basis for procedural design. But Intrusion Detection and Response using Mobile Agent Technology is not a typical application software because it doesn't have a main( ) function or starting point as it is present in typical application software so it is almost impossible to describe it in procedural form.

The design activity is completed here and can easily be mapped in coding or implementation section which is last activity of the software development process.

3.4 Object-Oriented Design

It includes a process of object-oriented decomposition and a notation for representing logical and physical as well as static and dynamic models of the system under design.

The four layers of object-oriented design pyramid are:

- **The Subsystem Layer** contains a representation of each of the subsystems that enable the software to achieve its customer-defined requirements and to implement the technical infrastructure that supports customer requirements. The subsystem design is derived by considering overall customer requirements (represented with use-cases) and the events and states that are externally observable (the object-behavior model).
- **The Class and Object Layer** contains the class hierarchies that enable the system to be created using generalizations and increasingly more targeted specializations. This layer also contains representations of each object. Class and object design is mapped from the description of attributes, operations, and collaborations contained in the CRC model.
- **The Message Layer** contains the design details that enable each object to communicate with its collaborators. This layer establishes the external and internal interfaces for the system. Message design is driven by the object-relationship model.
- **The Responsibilities Layer** contains the data structure and algorithmic design for all attributes and operations for each object. Responsibilities design is derived using the attributes, operations, and collaborations described in the CRC model.

The design pyramid focuses exclusively on the design of a specific product or system. It should be noted, however that another "layer" of design exists, and this layer forms the foundation on which the pyramid rests. The foundation layer focuses on the design of *domain objects* (called *design patterns*). Domain objects play a key role in building the infrastructure for the OO system by providing support for human/computer interface activities, task management, and data management. Domain objects can also be used to flesh out the design of the application itself.

3.5 Design Patterns

The best engineers in any field have an uncanny ability to see patterns that characterize a problem and corresponding patterns that can be combined to create a solution. Throughout the OOD process, a software engineer should look for every opportunity to reuse existing design patterns (when they meet the needs of the design) rather than creating new ones.

3.5.1 Describing a Design Patterns

All design patterns can be described by specifying the following information:

- The name of the pattern
- The intent of the pattern
- The "design forces" that motivate the pattern
- The solution that mitigates these forces
- The classes that are required to implement the solution
- The responsibilities and collaboration among solution classes
- Guidance that leads to effective implementation
- Example source code or source code templates
- Cross-references to related design patterns

The design pattern name is itself an abstraction that conveys significant meaning. once the applicability and intent are understood. *Design forces* describe the data, functional, or behavioral requirements associated with part of the software for which the pattern is to be applied. In addition forces define the constraints that may restrict the manner in which the design is to be derived. In essence, design forces describe the environment and conditions that must exist to make the design pattern applicable. The pattern characteristics (classes, responsibilities, and collaborations) indicate the attributes of the design that may be adjusted to enable the pattern to accommodate a variety of problems. These attributes represent characteristics of the design that can be searched (e.g. via a database) so that an appropriate pattern can be found. Finally, guidance associated with the use of a design pattern provides an indication of the ramifications of design decisions.

3.5.2 Using Patterns in Design

In an object-oriented system, design patterns can be used by applying two different mechanisms: inheritance and composition. Using *inheritance*, an existing design pattern becomes a template for new subclass. The attributes and operations that exist in the pattern become part of the subclass.

Composition is a concept that leads to aggregate objects. That is, a problem may require objects that have complex functionality (in the extreme, a subsystem accomplishes this). The complex object can be assembled by selecting a set of design patterns and composing the appropriate object (or subsystem). Each design pattern is treated as a black box, and communicates among the patterns occurs only via well defined interfaces.

3.6 Object-Oriented Design Process

UML design modeling addresses the structural model, behavioral model, implementation model, and environmental model views.

3.6.1 Structural Model

Data and functionality are viewed from inside the system. That is, static structure (classes, objects, and relationships) is modeled.

3.6.2 What is a Class?

A *class* is something that encapsulates information and behavior. Traditionally we've approached systems with the idea that we have the information over here on the database side, and the behavior over there on the application side. One of the differences with the object-oriented approach is the joining of a little bit of information with the behavior that affects the information. We take a little bit of information and a little bit of behavior, and encapsulate them into something called a class.

3.6.3 Finding a Class

A good place to start when finding classes is the flow of events for the use-cases. Looking at the nouns in the flow of events will let us know what some of the classes are. When looking at the nouns, they will be one of four things:

- An actor
- A class
- An attribute of a class
- An expression that is not an actor , class, or attribute

By filtering out all of the nouns except for the classes, we have found classes identified for our system.

3.6.3.1 Class Diagram

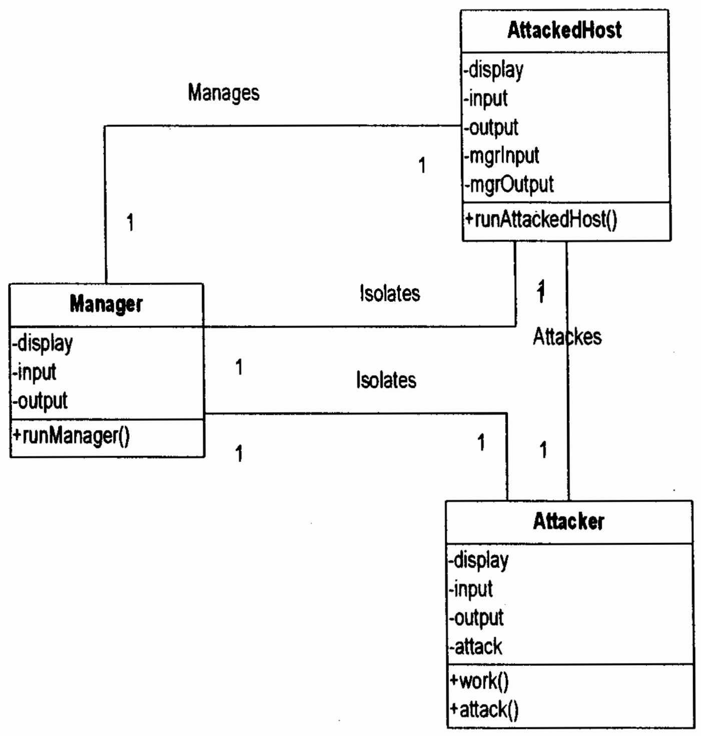


Figure 3.6 class diagram of intrusion detection and response using mobile gent technology

A *Class diagram* in Figure3.6 is used to display some of the classes and packages of classes in the system. It gives a static picture of the pieces in the system, and of the relationships between them.

3.6.4 Behavioral Model

This part of the analysis model represents the dynamic or behavioral aspects of the system. It also depicts the interactions or collaborations between various structural elements described in the user model and structural model views.

3.6.4.1 Interaction Diagrams

An *Interaction diagram* shows, step-by-step, one of the flows through a use-case. There are two types of Interaction diagrams:

- **Sequence Diagram** represents dynamic behavior which is time oriented. It can show the focus of control.
- **Collaboration Diagram** represents dynamic behavior which is message oriented. It can show the data flow.

3.6.4.2 Sequence Diagram

A Sequence diagram shown in Figure 3.7 is an interaction diagram, which is ordered by time; it is read form the top to the bottom.

We can read this diagram by looking at the objects and messages. The objects that participate in the flow are shown in rectangles across the top of the diagram.

The actor objects, involved in the use-case are also shown in the diagram.

Each object has a *lifeline*, drawn as a vertical dashed line below the object. A message is drawn between the lifelines of two objects to show that the objects communicate. Each message represents one object making a function call of another. Messages can also be reflexive, showing that an object is calling one of its own operations.

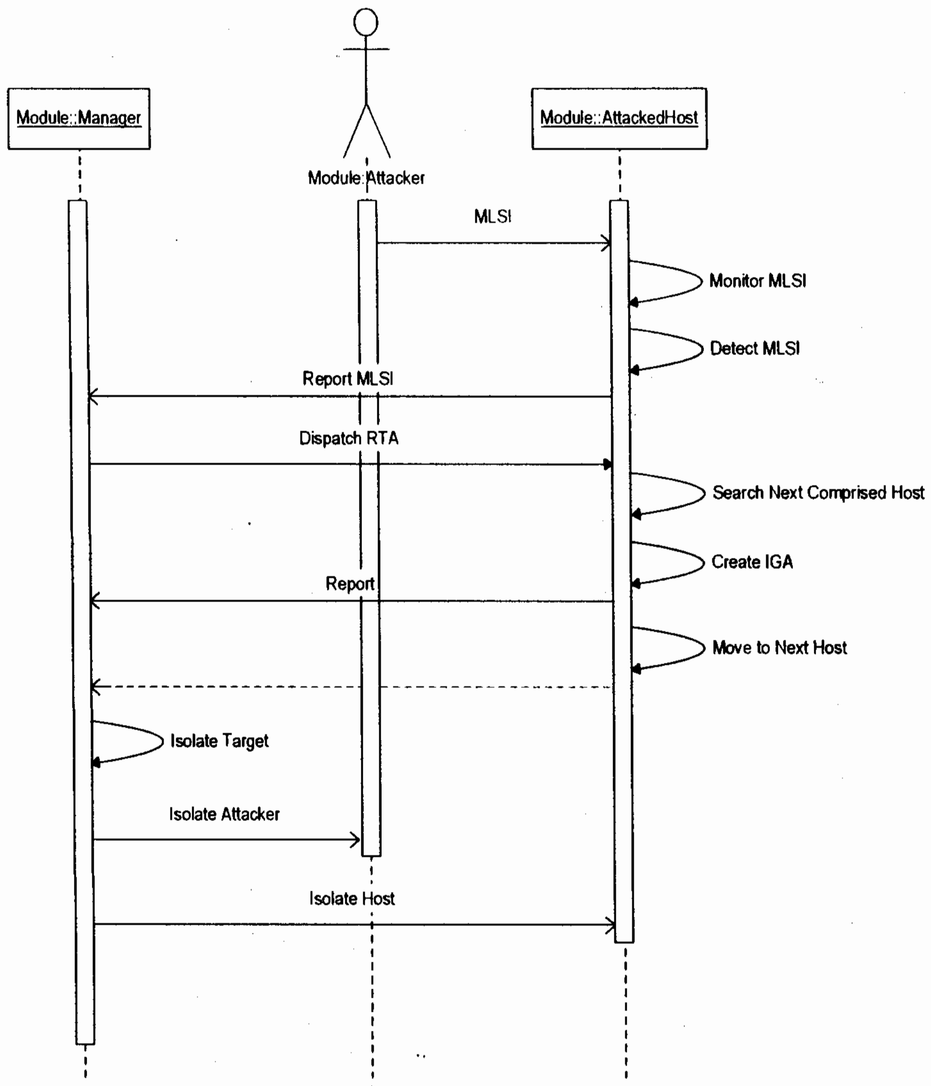


Figure 3.7 Sequence Diagram for intrusion detection and response using mobile agent technology

3.6.4.3 State Transition Diagrams

State Transition diagrams in Figures 3.8, 3.9 and 3.10 shows the life cycle of a single object, from the time it is created until it is destroyed.

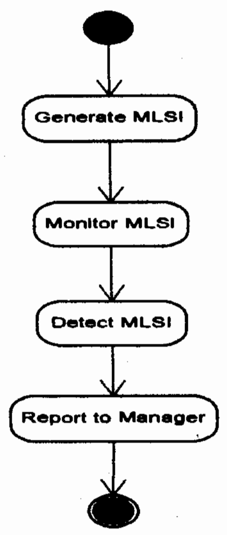


Figure 3.8 state transition diagram for the use case Generate MLSI

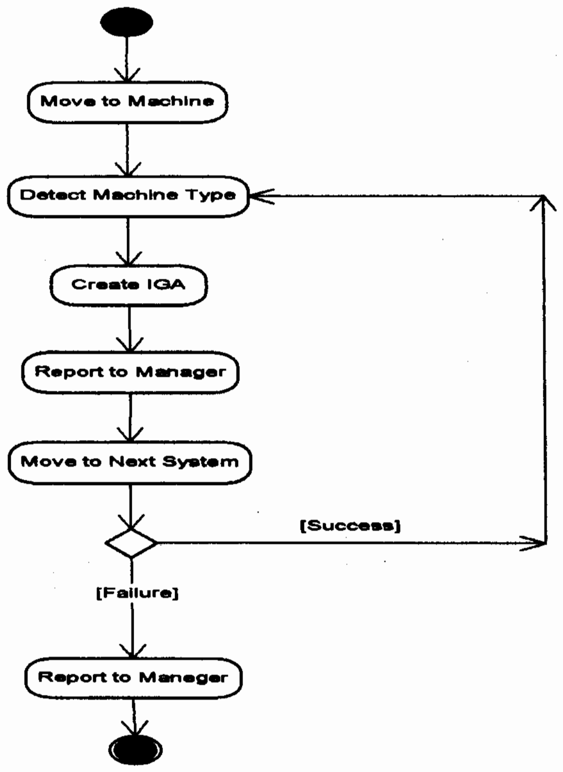


Figure 3.9: State Transition Diagram for Use-Case Dispatch RTA

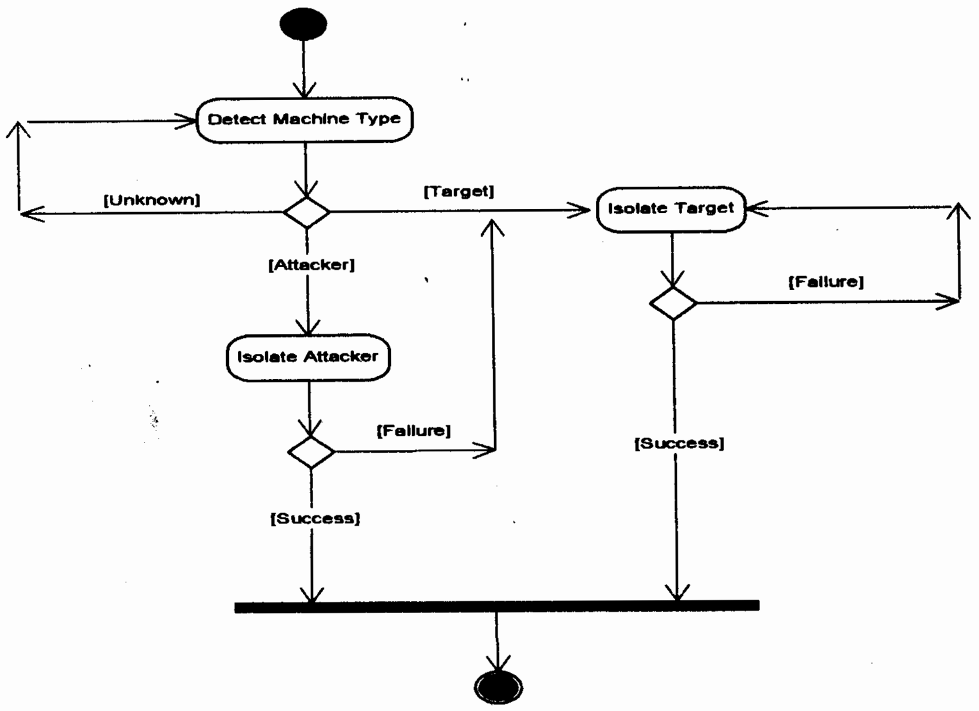


Figure 3.10: State Transition Diagram for Use-Case Isolate Machine

Chapter 4

Implementation

4 Implementation

This is very important phase in the software engineering pyradigm because no matter how efficiently analysis has been done? or how brilliantly the design has been prepared? Although programming is an outgrowth of analysis and design, all the programming and implementation skills have to be applied here, because any inefficiency on part of the programmer will hammer the quality of the software. Another important aspect of this phase is that, although this phase is succeeded by the testing phase, but during the implementaion phase the programmer is best equiped for glass box testing of the software, because at this stage he has the access to the code.

4.1 Implementation Techniques

The following techniques are used during the implementation of our project.

4.1.1 Object-Oriented Programming

Although all areas of object technologies have received significant attention within the software community, no subject has produced more books, more discussion, and more debate than *object-oriented programming* (OOP). Hundreds of books have been wirtten on C++ and Java programming, and hundreds more are dedicated to less widely used OO languages.

The software engineering viewpoint stresses OOA and OOD and considers OOP (coding) an important, but secondary activity that is an outgrowth of analysis and design. The reason for this is simple. As the complexity of systems increases, the design architecture of the end product has a significantly stronger influence on its success than the programming language that has been used. And yet, "language wars" continue to rage.

4.1.2 Agent Based Programming

In the software engineering context, reuse is an idea both old and new programmers have stressed upon, since the earliest days of computing, but the early approach to reuse was ad hoc. Today, complex, high-quality computer-based systems must be built in very short time periods. This mitigates toward a more organized approach to reuse.

Computer-based software engineering (CBSE) is a process that emphasizes the design and construction of computer-based systems using reusable software "components."

Agents are task oriented objects. When we want to move agents(Mobile agents), we use such criteria, that, with minimum amount of code, we can do enough work and be able to complete our task.Actually our emphasis is on creating lightweight mobile agents hence we are using such a tool in which migration of agents across the plateforms is easiest.

4.2 Implementation Tools

Our software is developed using three tools D Agents, TCL , and java. The reason for selecting these tools is that some work in the background on the system level is done in TCL and as our work is with Agents ,not simply agents but mobile agents and for the mobilty of agents we have to use some tool and the tool we have selected is D Agents. D Agents is an interpreted version of the interpretor of TCL. In the following text we describe the main reasons of selecting these tools.

4.2.1 D Agents

D Agent is a powerful tool for the rapid development of complex agents that run on Unix(family) workstations.It is an effective plateform for developing mobile agents and for the development of small to medium sized applications. As D Agents are developed in a scripting language TCL , so D Agent's agents can use all of the standard TCL commands as well as special set of commands that are specifically designed as a TCL extension(D Agents). These special set of commands allow an agent to migrate an agent from one machine to another, to create child agents, to communicate with some other agents, and to obtain the agent's current network location.In addition,D Agents, like all TCL based systems, can be extended with user defined commands to create a more powerful agent system—for example, a set of text processing commands can be made available to all agents at a particular site. This provides developers with the ability to come up with services for users which can be easily accessed by the agents.

4.2.1.1 Agent Movement

The real power of agents comes with the ability to move from machine to machine. In D Agents, this migration is accomplished with a single command *agent_jump* . Which can appear anywhere in an agent. It captures the current state of the agent and transfers this image to the server on the destination machine. The server restores the state image and the agent continues the execution from the command immediately after the *agent_jump*. In other words *agent_jump*, allows an agent to suspend its execution at an arbitrary point, transport to another machine, and resume execution on the new machine at the exact point at which it left off. Once an D Agent's agent has migrated to the machine, it can access resources and communicate with other agents on that machine. Once it finishes its local task, it can migrate to the next machine.

4.2.1.2 Agent Communication

Once an agent is at particular place,it goes about its tasks by communicating with other agent at the site. There are two ways for agents to communicate with each other. The first is message passing, which uses the traditional send and receive concepts. The *agent_send* command sends the message to another agent and *agent_receive* command receives the incomming message. The second form of the communication is a direct connection, which is essentially a message stream. An agent establishes a direct connection with an other agent using the *agent_meet* command. The two agents then exchange the messages over the connection.

4.2.2 TCL

TCL is a string-based command language. The language has only a few fundamental constructs and relatively little syntax, which makes it easy to learn. The basic mechanisms are all related to strings and string substitutions, so it is fairly easy to visualize what is going on in the interpreter. The model is a little different than some other languages we already familiar with.

TCL is high level scripting language. It has enjoyed enormous popularity in the UNIX community. As TCL is interpreted, so it is highly portable and easier to make secure. It can be embedded in other applications, which allows these applications to implement part of their functionality with mobile TCL agents. TCL can be extended with user defined commands. Which makes it easy to tightly integrate agent functionality with rest of the language and allows a resource to provide a package of TCL commands that an agent uses to access the resource. A package of TCL commands is more efficient than encapsulating the resource within an agent and is an attractive alternative in certain application

4.2.2.1 TCL Script as Agents

A D Agents's agent is a TCL script that run on the top of the modified interpreter and a TCL extension. The modified interpreter provides the explicit stack and the state capture routines. The extension provides the set of commands that the script uses to migrate , communicate, and create child agents.
The most important commands are

- *agent_begin*
- *agent_submit*
- *agent_jump*
- *agent_name*
- *agent_send*
- *agent_receive*
- *agent_ meet*
- *agent_accept*
- *agent_end*

An agent uses the *agent_begin* command to register with the server and obtain an identifier in the flat namespace. An identifier currently consists of the IP address of the server, a unique integer, and an optional symbolic name that the agent specifies later with the *agent_name* command. The *agent_submit* command is used to create a child agent on a particular machine. The *agent_jump* command migrates an agent to a particular machine. The command captures the internal state of the agent, packages the state image for transport, and sends the state image to the destination server. The server retrieves the state image, selects a new identifier for the agent and starts a TCL interpreter. The TCL interpreter restores the state image and resumes agent execution at the statement immediately after the *agent_jump*.

The *agent_send* and *agent_receive* commands are used to send and receive messages. The *agent_meet* and *agent_accept* commands are used to establish a direct connection between agents. For direct connections, the source agent uses *agent_meet* to send a connection request to the destination agent. The destination agent uses *agent_accept* to receive the connection request and send either an acceptance or rejection.

4.3 Implemetation of Some Important Functionalities

4.3.1 Monitoring Agent

4.3.1.1 Main Module

```
/* get manager machine's IP address and owner and user and group */
  if ((HOME_DIR = util.getTrainds()) == NULL) {
     log->write("unable to get home directory.");
     exit(-1);
  }


  user      = util.getIdaEnvValue("USER");
  log->write("%s\n", user);
  group     = util.getIdaEnvValue("GROUP");
  log->write("%s\n", group);
  managerIP  = util.getIdaEnvValue("MANAGER_IP");
  log->write("%s\n", managerIP);
  managerPORT = util.getIdaEnvValue("MANAGER_PORT");
  log->write("%s\n", managerPORT);
```

```
myIP      = util.getIPAddress(util.getHostname());
log->write("%s\n", myIP);
owner     = util.getIdaEnvValue("OWNER");
log->write("%s\n", owner);

if (managerIP == NULL || managerPORT == NULL || owner == NULL ||
      user == NULL || group == NULL || myIP == NULL) {
      .     .     .     .

      .     .     .     .

      .     .     .     .

      .     .     .     .
      log->write("unable to get default value from config file.");
   return(ERR_SETUP_MA);
}

/* creat Tcpip instance */
if ((tcpip = new Tcpip(managerIP, (u_short)atoi(managerPORT))) == NULL) {
   log->write("unable to create Tcpip instance.");
   return(ERR_SETUP_MA);
}

/* change uid gid */
if (util.switchGroupId(group) < 0) {
   log->write("unable to switch group %s.", group);
   return(ERR_SETUP_MA);
}
if (util.switchUserId(user) < 0) {
   log->write("unable to switch user %s.", user);
   return(ERR_SETUP_MA);
}
if ((HOME_DIR = util.getHomeDir(user)) == NULL) {
    log->write("unable to get user %s's home directory.", user);
    return(ERR_SETUP_MA);
  }

  /* change HOME directory */
  if ((HOME = new char[strlen(HOME_DIR) + 7]) == NULL)
    return(ERR_SETUP_MA);
  memset(HOME, '\0', strlen (HOME_DIR) + 7);
  strcpy(HOME, homes);
  strcat(HOME, HOME_DIR);

  if (putenv(HOME) != 0) {
    log->write("unable set environment variable \"LANG\".");
      return(ERR_SETUP_MA);
  }

  /* main loop */
  for (;;) {
    if ((buf = new char[BUF_MAX + BUF_MAX]) == NULL)
        goto END;
      memset(buf, '\0', BUF_MAX+BUF_MAX);

      /* read one filtering log */
      if ((read(fp, fingerPrint, BUF_MAX)) <= 0)
      goto END;

      /* setup client of socket connection */
      i = 0;
      while ((tcpip->setupClient() != SUCCESS) && limiter >= i) {
        if (i >= MAX_REPEAT_TIME) goto END;
```

```
            if (debug == 0) {
               if (i > 0) {
                    log->write("ERROR: Unable to connect to manager machine.\n");
                    log->write("ERROR: After %d sec, Connection request will be sen to
manager ,achine\n", intervalTime[i]);
                    }
            }

            i++;
            sleep(intervalTime[i]);
          }

                    END:
          /* clean up memory */
          fingerPrint[0] = '\0';
          if (buf != NULL) delete [] buf;
          buf = NULL;
   }
   delete(tcpip);
}

for (;;) {
      if ((read(fd, mlsi, BUF_MAX)) <= 0)
          continue;
#ifdef DEBUG
          fprintf(stdout, "mlsi = %s\n", mlsi);
#endif
      if (sscanf(mlsi, "%s %s %s %s %s %d",
                    uid, pid, session_id, audit_id, audit_date, &mlsiflag) < 0)
            continue;
#ifdef DEBUG
```

```
        fprintf(stdout, "%s %s %s %s %s %d\n", uid, pid, audit_id, session_id,
audit_date, mlsiflag);
#endif

        plugin_load(sharedMemory, dtRuleDB,
                        uid, pid, audit_id, session_id, audit_date, mlsiflag,
                        managerIP, managerPORT, myIP);
   }
}
```

4.3.1.2 Report of Monitoring Agent to Manager

```
/* send data to manager machine */
        res = sscanf(fingerPrint, "%s %s %s %1024c",
                        uid, pid, session_id, audit_date);
        if (res < 0) {
           tcpip->closeConnect();
           goto END;
        }

        res = sprintf(buf, "%s %s %s %s %s %s",
                        uid, pid, myIP, session_id, audit_id, audit_date);
        if (res < 0) {
           tcpip->closeConnect();
           goto END;
        }
```

4.3.2 Information Gathering Agent

4.3.2.1 Creating Informatin Gathering Agent

Route Tracing Agent will use the follwing to create Information Gathering Agent.

```
# catch any error
```

```
if {[catch {
        agent_submit  $agent (local-ip) –vars machine –procs iga –script { iga $machine}
        .    .    .    .    .
        .    .    .    .    .
        .    .    .    .    .
        .    .    .    .    .
        .    .    .    .    .
        agent_end
        }error_message]} then {
        agent_end.
        }
```

4.3.3 Route Tracing Agent

4.3.3.1 Moving Route Tracing to a machine

Agent_jump machine /*machine is the variable for which Monitoring agent will provide the value for the first time and then Route Tracing Agent will itself detect where to go and the ip address of machine will be stored in the variable machine. */

4.3.3.2 Exporting time stamp

```
sscanf(fl, "%*s %s %s %s %s", month, day, hms, year);
  for (i = 0; i < 12; i++)
        if (strcmp(month, MONTHS[i]) == 0)
           break;
  dt = (char *)malloc(21);
  sprintf(dt, "%s/%02d/%s--%s", year, ++i, day, hms);
  return(dt);
}
```

4.3.4 Working with log files

4.3.4.1 Initializing a log file

```
void init_loglist()
{
   int i;

   for (i = 0; i <= RETURN; i++) {
         loglist[i] = (LOGLIST *)malloc(sizeof(LOGLIST));
         loglist[i]->sym = NULL;
         loglist[i]->next = NULL;
   }
}
```

4.3.4.2 Reset log file

```
for (i = 0; i <= RETURN; i++) {
         free(loglist[i]->sym);
         for (p = loglist[i]->next; p != NULL; p = q) {
            q = p->next;
            free(p->sym);
            free(p);
         }
         loglist[i]->sym = NULL;
         loglist[i]->next = NULL;
   }
}
```

4.3.4.3 Set log file

```
int set_log_list(au_token_t *tok, int type, char *delim, int retopt)
{
   char default_delim[] = ",";
   char *d;
   char *hoge;
   int  i;
   char fld[BUF_LENGTH];
   LOGLIST *p;
```

```
char time[BUF_LENGTH];

/* define delimitator */
if (delim == NULL) d = default_delim;
else d = delim;

switch (tok->id)
{
        /* AU_ARG */
case (0x2D):
        for (p = loglist[ARG]; p->next != NULL; p = p->next)
            ;
        sprintf(fld, "%d", (unsigned int)tok->un.arg.val);
        p->sym = (char *)malloc(strlen(fld) + 1);
        strcpy(p->sym, fld);
        p->next = (LOGLIST *)malloc(sizeof(LOGLIST));
        p = p->next;
        p->sym = NULL;
        p->next = NULL;
        break;

        /* AU_ATTR */
case (0x31):
        sprintf(fld, "%o", (unsigned int)tok->un.attr.mode);
        loglist[FMOD]->sym = (char *)malloc(strlen(fld) + 1);
        strcpy(loglist[FMOD]->sym, fld);

        strcpy(fld, au_uid_to_uname((unsigned int)tok->un.attr.uid, type));
        loglist[A_UID]->sym = (char *)malloc(strlen(fld) + 1);
        strcpy(loglist[A_UID]->sym, fld);
        strcpy(fld, au_gid_to_gname((unsigned int)tok->un.attr.gid, type));
        loglist[A_GID]->sym = (char *)malloc(strlen(fld) + 1);
        strcpy(loglist[A_GID]->sym, fld);
        break;

        /* AU_EXIT */
    case (0x52):
        sprintf(fld, "%d", (unsigned int)tok->un.exit.status);
        loglist[EXIT]->sym = (char *)malloc(strlen(fld) + 1);
        strcpy(loglist[EXIT]->sym, fld);
        break;

        /* AU_HEADER */
    case (0x14):
        strcpy(fld, au_event_to_char(tok->un.header.event, type));
        loglist[EVENT]->sym = (char *)malloc(strlen(fld) + 1);
        strcpy(loglist[EVENT]->sym, fld);

        strcpy(fld, au_event_to_class(tok->un.header.event, type));
        set_class_loglist(fld);

        strcpy(fld, au_time_to_char(tok->un.header.time, type));
        hoge = export_timestamp(fld);
        strcpy(time, hoge);
        free(hoge);
        loglist[TIME]->sym = (char *)malloc(strlen(time) + 1);
        strcpy(loglist[TIME]->sym, time);
        break;

        /* AU_PATH */
    case (0x23):
        loglist[PATH]->sym = (char *)malloc(strlen(tok->un.path.name) + 1);
        strcpy(loglist[PATH]->sym, tok->un.path.name);
        break;

        /* AU_RETURN */
    case (0x27):
        strcpy(fld, au_error_to_char(tok->un.ret.retval, type));
        loglist[RETURN]->sym = (char *)malloc(strlen(fld) + 1);
        strcpy(loglist[RETURN]->sym, fld);
        break;

        /* AU_SUBJECT */
    case (0x24):
        sprintf(fld, "%s", au_uid_to_uname(tok->un.subj.auid, type));
        loglist[AUDITID]->sym = (char *)malloc(strlen(fld) + 1);
        strcpy(loglist[AUDITID]->sym, fld);

        sprintf(fld, "%d", tok->un.subj.sid);
        loglist[SESSIONID]->sym = (char *)malloc(strlen(fld) + 1);
        strcpy(loglist[SESSIONID]->sym, fld);

        strcpy(fld, au_uid_to_uname(tok->un.subj.euid, type));
        loglist[EUID]->sym = (char *)malloc(strlen(fld) + 1);
        strcpy(loglist[EUID]->sym, fld);

        strcpy(fld, au_uid_to_uname(tok->un.subj.ruid, type));
        loglist[S_UID]->sym = (char *)malloc(strlen(fld) + 1);
        strcpy(loglist[S_UID]->sym, fld);

        strcpy(fld, au_gid_to_gname(tok->un.subj.egid, type));
        loglist[EGID]->sym = (char *)malloc(strlen(fld) + 1);
```

```
strcpy(loglist[EGID]->sym, fld);

strcpy(fld, au_gid_to_gname(tok->un.subj.rgid, type));
loglist[S_GID]->sym = (char *)malloc(strlen(fld) + 1);
strcpy(loglist[S_GID]->sym, fld);

sprintf(fld, "%d", tok->un.subj.pid);
loglist[PID]->sym = (char *)malloc(strlen(fld) + 1);
strcpy(loglist[PID]->sym, fld);

if (tok->un.subj.tid.driver == 0) {
   loglist[TTY]->sym = (char *)malloc(8);
   strcpy(loglist[TTY]->sym, "console");
} else {
   sprintf(fld, "%d", tok->un.subj.tid.driver);
   loglist[TTY]->sym = (char *)malloc(strlen(fld) + 1);
   strcpy(loglist[TTY]->sym, fld);
}
break;

 case (0x3C):
for (p = loglist[EXEC_ARG]; p->next != NULL; p = p->next)
   ;
p->sym = (char *)malloc(strlen(tok->un.exec_arg.args[0]) + 1);
strcpy(p->sym, tok->un.exec_arg.args[0]);
for (i = 1; i < tok->un.exec_arg.num; i++) {
   p->next = (LOGLIST *)malloc(sizeof(LOGLIST));
   p = p->next;
   p->sym = (char *)malloc(strlen(tok->un.exec_arg.args[i]) + 1);
   strcpy(p->sym, tok->un.exec_arg.args[i]);
   p->next = NULL;
```

```
        }

        /* AU_TEXT */
    case (0x28):
        for (p = loglist[TEXT]; p->next != NULL; p = p->next)
            ;
        p->sym = (char *)malloc(strlen(tok->un.text.data) + 1);
        strcpy(p->sym, tok->un.text.data);
        p->next = (LOGLIST *)malloc(sizeof(LOGLIST));
        p = p->next;
        p->sym = NULL;
        p->next = NULL;
        break;

        /* AU_SOCKET */
    case (0x2E):
        sprintf(fld, "%d", tok->un.socket.lport);
        loglist[L_PORT]->sym = (char *)malloc(strlen(fld) + 1);
        strcpy(loglist[L_PORT]->sym, fld);

        strcpy(fld, au_ip_to_char(tok->un.socket.laddr));
        loglist[L_ADDR]->sym = (char *)malloc(strlen(fld) + 1);
        strcpy(loglist[L_ADDR]->sym, fld);

        sprintf(fld, "%d", tok->un.socket.fport);
        loglist[R_PORT]->sym = (char *)malloc(strlen(fld) + 1);
        strcpy(loglist[R_PORT]->sym, fld);

        strcpy(fld, au_ip_to_char(tok->un.socket.faddr));
        loglist[R_ADDR]->sym = (char *)malloc(strlen(fld) + 1);
        strcpy(loglist[R_ADDR]->sym, fld);
        break;

        /* AU_COMM */
    case (0x71):
        sprintf(fld, "%d", tok->un.parent.pid);
        loglist[PPID]->sym = (char *)malloc(strlen(fld) + 1);
        strcpy(loglist[PPID]->sym, fld);
        break;

    default:
        break;
    }
    return (SUCC);
}
```

4.3.4.4 Write to Log

```
void Logging::write(const char * str, ...)
{
    va_list param;
    char disp[4096];
    char *tmp;
    char timestr[1024];

    va_start(param, str);
    vsnprintf(disp, 4095, str, param);

    while ((tmp = (char *)strchr(disp, '\n')))
        tmp[0]=' ';
    if (disp[strlen(disp)-1] == '\n')
        disp[strlen(disp)-1] = 0;
    if (log) {
            time_t t;


            t = time(NULL);
            tmp = ctime(&t);


            strcpy(timestr, tmp);
            timestr[strlen(timestr)-1] = 0;
            .        .        .        .

            .        .        .        .

            .        .        .
            fprintf(log, "[%s][%d] %s\n", timestr, getpid(), disp);
            fflush(log);
    } else {
            syslog(LOG_NOTICE, disp);
    }
    va_end(param);
}

if ((type = checkType(line)) == ERR_LOG_UNKNOWN) return(NULL);
    token->type = type;

    switch (type) {

    case LOG_SESSION_ID:
        strcpy (token->session.arg, "");
            res = sscanf(line, "%s %s %d %s %s %s %s %d %d %s %s %d %s %1024c",
                        token->session.header, token->session.bsmid,
                        &(token->session.session_id), token->session.uid,
                        token->session.gid, token->session.ruid,
                        token->session.rgid, &(token->session.pid),
                        &(token->session.ppid), token->session.date,
                    token->session.tty, &(token->session.mode),
                    token->session.cmd, token->session.arg);

        if (res < 0 ) return(NULL);
        return(token);

    case LOG_OPEN:
      res = sscanf(line, "%s %s %d %s %s %s %s %d %s %s %s %s %d",
                    token->open.header, token->open.bsmid,
                    &(token->open.session_id), token->open.uid,
                    token->open.gid, token->open.ruid, token->open.rgid,
                    &(token->open.pid), token->open.date, token->open.path,
                    token->open.owner, token->open.owner_group,
                    &(token->open.mode));

        if (res < 0) return(NULL);
        return(token);

    case LOG_CREAT:
      res = sscanf(line, "%s %s %d %s %s %s %s %d %s %s",
                    token->creat.header, token->creat.bsmid,
                    &(token->creat.session_id), token->creat.uid,
                    token->creat.gid, token->creat.ruid,
                    token->creat.rgid, &(token->creat.pid),
                    token->creat.date, token->creat.path);

        if (res < 0) return (NULL);
        return (token);

    case LOG_SYMLINK:
      res = sscanf(line, "%s %s %d %s %s %s %s %d %s %s %s",
                token->symlink.header, token->symlink.bsmid,
                &(token->symlink.session_id), token->symlink.uid,
                token->symlink.gid, token->symlink.ruid,
                token->symlink.rgid, &(token->symlink.pid),
                token->symlink.date, token->symlink.text,
                token->symlink.path);

        if (res < 0) return(NULL);
    return(token);

    case LOG_CH_ATTR:
      res = sscanf(line, "%s %s %d %s %s %s %s %d %s %s %s",
                token->ch_attr.header, token->ch_attr.bsmid,
                &(token->ch_attr.session_id), token->ch_attr.uid,
                token->ch_attr.gid, token->ch_attr.ruid,
                token->ch_attr.rgid, &(token->ch_attr.pid),
                token->ch_attr.date, token->ch_attr.owner,
                token->ch_attr.path);

        free(line);
        if (res < 0) return(NULL);
        return(token);

    case LOG_SESSION:
      strcpy(token->sessionT.header, strtok(line, ","));
        strcpy(token->sessionT.bsmid,  strtok(NULL, ","));
        token->sessionT.session_id = atoi(strtok(NULL, ","));
        strcpy(token->sessionT.uid, strtok(NULL, ","));
        token->sessionT.pid = atoi(strtok(NULL, ","));
        strcpy(token->sessionT.tty, strtok(NULL, ","));
        strcpy(token->sessionT.date, strtok(NULL, ","));
        return(token);

    case LOG_CONNECTION:
      strcpy(token->connection.header, strtok(line, ","));
        strcpy(token->connection.type, strtok(NULL, ","));
        strcpy(token->connection.bsmid, strtok (NULL, ","));
        token->connection.session_id = atoi(strtok(NULL, ","));
        strcpy(token->connection.uid, strtok(NULL, ","));
        token->connection.pid = atoi(strtok(NULL, ","));
        strcpy(token->connection.tty, strtok(NULL, ","));
        strcpy(token->connection.date, strtok(NULL, ","));
        return(token);

    case LOG_LSOF:
      break;

    default:
      return(NULL);
    }
    return(NULL);
}
```

4.3.5 Manager

```
#ifdef DEBUG
  fprintf(stdout, "log = {\n%s}\n", log);
  fprintf(stdout, "intrusion type = %c\n", *log);
#endif
  fprintf(stdout, "!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!\n");
  switch(*log) {
  case '0':
    fprintf(stdout, "!!  Root privilege maybe gained            !!\n");
```

```
        break;
    case '1':
      fprintf(stdout, "!!  File maybe overwrited                    !!\n");
        break;
    case '2':
      fprintf(stdout, "!!  D-Analysis found intrusion               !!\n");
        break;
    default:
        error_no_exit("Illegal data received");
        break;
    }
    ipaddr = inet_ntoa(*host);
    fprintf(stdout, "!!  HOST = %25s  !!\n", ipaddr);
    fprintf(stdout, "!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!\n");

    if ((home_dir = getenv("HOME")) == NULL)
      error_exit("unable to get home directory name");
    memset(logfn, '\0', sizeof(logfn));
    tt = time(NULL);
    tp = localtime(&tt);
    strftime(logfn, sizeof(logfn), "%Y%m%d%H%M%S.log", tp);
    memset(logfile, '\0', sizeof(logfile));
    strncpy(logfile, home_dir, strlen(home_dir));
    strcat(logfile, "/log/");
    strcat(logfile, logfn);
    if ((fp = fopen(logfile, "w")) == NULL)
        error_no_exit("Cannot open file");
    else
        fprintf(stdout, "Check result ----> %s\n", logfile);

    for (i = 0; i < 65536; i++) {
        if (*log == '\0') break;
        fprintf(fp, "%c", *log++);
    }
    fclose(fp);
// setup signal
    signal(SIGTERM, (*signal_handler));
    signal(SIGKILL, (*signal_handler));
    signal(SIGSEGV, (*signal_handler));

    if ((childpid = fork()) < 0) {
        error_exit("unable to fork child process.\n");
        exit(1);
    } else if (childpid > 0) {
      signal(SIGTERM, signal_handler);
      signal(SIGKILL, signal_handler);
      signal(SIGPIPE, signal_handler);

      while (wait ((int *) 0) != childpid)
           ;

      exit(0);
    } else {
        signal(SIGTERM, signal_handler);
        signal(SIGKILL, signal_handler);
        signal(SIGPIPE, signal_handler);
        alert_manager();
        exit(0);
}
```

4.4 Isolating Source/Attacker

```
if ((tcpip->writeEncrypt(buf, key, pass, owner)) != SUCCESS) {
        if (*log == '\0') break;
        fprintf(fp, "%c", *log++);
    }
    fclose(fp);
// setup signal
    signal(SIGTERM, (*signal_handler));
    signal(SIGKILL, (*signal_handler));
    signal(SIGSEGV, (*signal_handler));

    if ((childpid = fork()) < 0) {
            error_exit("unable to fork child process.\n");
            exit(1);
    } else if (childpid > 0) {
        signal(SIGTERM, signal_handler);
        signal(SIGKILL, signal_handler);
        signal(SIGPIPE, signal_handler);

        while (wait ((int *) 0) != childpid)
                ;

        exit(0);
    } else {
            signal(SIGTERM, signal_handler);
            signal(SIGKILL, signal_handler);
            signal(SIGPIPE, signal_handler);
            alert_manager();
            exit(0);
  }
```

4.4 Isolating Source/Target

```
if ((tcpip->writeEncrypt(buf, key, pass, owner)) != SUCCESS) {
    tcpip->closeConnect();
    goto END;
}
/* close connection */
tcpip->closeConnect();
```

Chapter 5

Testing

5 Testing

The overall objective of the testing process is to identify the maximum number of errors in the code with a minimum amount of efforts. Finding an error is thus considered a success rather than failure. On finding an error, efforts are made to correct it.

5.1 Testing Process

Test consists of a number of test cases, where different aspects of the part of the project under test are checked. Each test case tells what to do, what data to use, and what results to expect. When conducting the test, the results including deviations from the planned test cases are not in a test protocol. Normally a deviation indicates an error in the system (although some times the test case is wrong, and the system is right). An error is noted and described in a test report for removal or directly removed by the programmer who developed that part.

5.2 General Types of Errors

Error can be of following types:

1. Functional error (e.g. function is not working correctly or missing).
2. Non-Functional error (e.g. performance is slow)
3. Logical error (e.g. error in algorithm, user interface errors is not considered as a logical error).

5.3 Testing Strategies

A strategy for software testing may be viewed as the spiral (Figure 5.1). Unit testing begins at the vortex of the spiral and concentrates on each unit (i.e., component) of the software as implemented in source code. Testing progresses by moving outward along the spiral to integration testing, where the focus is on design and the construction of the software architecture. Taking another turn outward on the spiral, we encounter validation testing, where requirements established as part of software requirements analysis are validated against the software that has been constructed. Finally, we arrive at system testing, where the software and other system elements are tested as a whole. To test computer software, we spiral out along stream-lines that broaden the scope of testing with each turn.

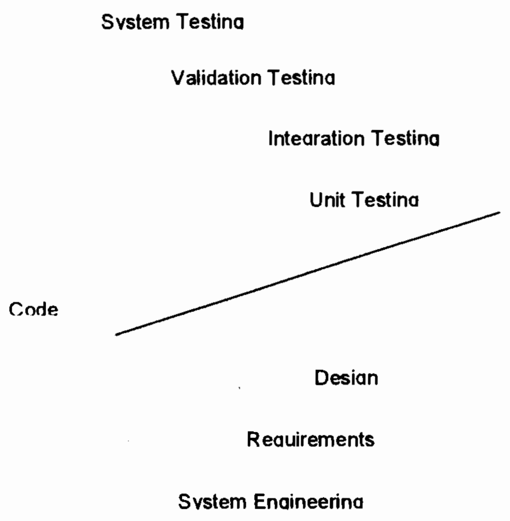


Figure 5.1: Testing Strategy

5.3.1 Unit Testing

Unit testing focuses verification effort on the smallest unit of software design—the software component of module. Using the component-level design description as guide, important control paths are tested to uncover errors within the boundary of the module. The relative complexity of tests is uncovered errors is limited by the constrained scope established for unit testing. The unit test is white-box oriented, and the step can be conducted in parallel for multiple components.

5.3.2 Integration Testing

Integration testing is a systematic technique for constructing the program structure while at the same time conducting tests to uncover errors associated with interfacing. The objective is to take unit tested components and build a program structure that has been dictated by design.

5.3.2.1 Top-down Integration

Top-down integration testing is an incremental approach for construction of program structure. Modules are integrated by moving downward through the control hierarchy, beginning with the main control module (main program). Modules subordinates (and ultimately subordinate) to the main control module control module are incorporated into the structure in either a depth-first or breath-first manner.

5.3.2.2 Bottom-up Integration

Bottom-up integration testing, as the name implies, begins construction and testing with atomic modules (i.e., components at the lowest levels in the program structure). Because components are integrated from the bottom up, processing required for components subordinate to a given level is always available and the need for stubs is eliminated.

5.3.2.3 Regression Testing

Each time a new module is added as part of integration testing, the software changes. New data flow paths are established, new I/O may occur, and new control logic is invoked. These changes may cause problems with functions that previously worked flawlessly. In the context of an integration test strategy, regression testing is the re-execution of some subset of tests that have already been conducted to ensure that changes have not propagated unintended side effects.

5.3.3 Validation Testing

Validation can be defined in many ways but a simple definition is that validation succeeds when software functions in a manner that can be reasonably expected by the customer.

5.3.4 System Testing

System testing is a series of different tests whose primary purpose is to fully exercise the computer-based system. Although each test has a different purpose, all work to verify that system elements have been properly integrated and perform allocated functions.

5.3.4.1 Security Testing

Security testing attempts to verify that protection mechanisms built into a system will, in fact, protect it from improper penetration.

5.3.4.2 Stress Testing

Stress testing executes a system in a manner that demands resources in abnormal quantity, frequency, or volume.

5.3.4.3 Performance Testing

Performance testing is designed to test the run-time performance of software within the context of an integrated system. Performance testing occurs throughout all steps in the testing process. Even at the unit level, the performance of an individual module may be assessed as white-box tests are conducted. However, it is not until all system elements are fully integrated that the true performance of a system can be ascertained.

5.4 Object-Oriented Testing Strategies

The classical strategy for testing computer software begins with "testing in the small" and works outward "testing in the large." We begin with unit testing, then progress towards integration testing, and culminate with validation and system testing. In conventional applications, unit testing focuses on the smallest program unit—the subprogram (e.g., modules, subroutine, procedure, component). Once each of these units has been tested individually, it is integrated into a program structure while a series of regression tests are run to uncover errors due to interfacing between the modules and side effects caused by the addition of new units. Finally, the system as a whole is tested to ensure that errors in requirements are uncovered.

5.4.1 Unit Testing in the OO Context

Class testing for OO software is equivalent to unit testing for conventional software. Unlike unit testing of conventional software, which tends to focus on the algorithmic detail of a module and the data that flow across the module interface, class testing for OO software is driven by the operations encapsulated by the class and the state behavior of the class.

5.4.2 Integration Testing in the OO Context

Because object-oriented software does not have a hierarchical control structure, conventional top-down and bottom-up integration strategies have little meaning. In addition, integrating operations one at a time into a class is often impossible because of the "direct and indirect interactions of the components that make up the class".
There are two different strategies for integration testing of OO systems:

5.4.2.1 Thread-based Testing

Thread-based testing, integrates the set of classes required to respond to one input or event for the system. Each thread is integrated and tested individually.

5.4.2.2 Use-based Testing

Use-based testing, begins the construction of the system by testing those classes (called independent classes) that use very few (if any) of server classes. After the independent classes are tested, the next layer of classes, called dependent classes that use the independent classes are tested. This layer of testing layers of dependent classes continues until the entire system is constructed.

5.4.2.3 Cluster Testing

Cluster testing is one step in the integration testing of OO software. Here, a cluster of collaborating classes (determined by examining the CRC and object-relationship model) is exercised by designing test cases that attempt to uncover errors in the collaborations.

5.4.3 Validation Testing in the OO Context

At the validation or system level, the details of class connections disappear. Like conventional validation, the validation of OO software focuses on user-visible actions and user-recognizable output from the system.

5.5 Test Plan

Test plan provides an overview of the testing effort for the product.

5.5.1 Intrusion Detection and Response Test Plan

- **Introduction**

  Our testing effort descriptions summarize IEEE 829-1983 for Software Test Documentation, which attempts to define a common set of test documents, to be used across the industry.

- **Test Items**
  1. Manager
  2. Monitoring Agent
  3. Route Tracing Agent
  4. Information Gathering Agent
  5. Isolating the Target
  6. Isolating the Source
- **Features to be tested**

  Our *Test Design Specification* will summarize the features to be tested

- **Features Not to be tested**

  1. Unit testing of the single modules can't be performed. Because a single module is not a stand-alone program. Stub software must be developed for each unit test.

  2. Comparison testing can't be performed for the items: *Isolating the target and isolating the attacker*. Because comparable software are not available.

3. Cyclomatic Complexity metric in the context of the basis path testing method can't be performed. Because the there are too many *independent paths*

- **Items Pass/Fail Criteria**

Item will be considered as pass if the test lead to an expected result or the behavior of the module is according to expectations. If otherwise the item will be considered as failed.

- **Suspension Criteria and Resumption Requirements**

If the test leads to a BSOD (Blue Screen of Death) or a severe error message from the operating system or a lethal software crash, further testing will be ceased. The test will be redone, incase the bug can't be reproduced.

- **Test Deliverables**

None

- **Environment Needs**

A Pentium® III or higher machine.

TCL 7.4 or higher

D Agents Ver 2.1 or higher

Sun Platform JDK for linux

Red-Hat Linux 7.2 (Family)

5.6 Test Design Specification

This specifies how a feature or group of features will be tested according to Standard 829.

5.6.1 Intrusion Detection and Response Test Design Specification

- **Features to be tested**

This specification includes

1. Stability of intrusion detection and response using mobile agent tech.
2. Validation of intrusion detection and response using mobile agent tech
3. User interface aspects of intrusion detection and response using mobile agent tech
4. Performance issues of intrusion detection and response using mobile agent tech
5. Scenarios and use-cases of intrusion detection and response using mobile agent tech
6. Stability of Monitoring Agent
7. Validation of Monitoring Agent
8. Performance issues of Monitoring Agent
9. Stability of Route Tracing Agent
10. Validation of Route Tracing Agent
11. Performance issues of Route Tracing Agent
12. Stability of Information Gathering Agent
13. Validation of Information Gathering Agent
14. Performance issues of Information Gathering Agent
15. Stability of Manager
16. Validation of Manager
17. Performance issues of Manager

- **Features not to be tested**

Our test case specification (section 5.7) defines each test associated with this design.

- **Feature Pass Fail Criteria**

A feature or a combination of features will be considered as pass if the tests lead to an expected result, or the behavior of the module is according to expectations. If otherwise the feature will be considered as failed.

5.7 Test case specification

This defines a test case. According to the standards 829, the test case specification includes the following sections.

5.7.1 Load test for Manager

Test Items

1. Manager

Input Specification

None

Output Specification

None

Environmental Needs

A Pentium® III or higher machine.

Red Hat Linux (Family)

Special Procedural requirements

None

Intercase Dependencies

None

5.7.2 Load Test for Target

Test Items

Target

Monitoring Agent

Input Specification

Attack from attack

Output Specification

None

Environmental Needs

A Pentium® III or higher machine.

Red Hat Linux (Family)

Special Procedural requirements

None

Intercase Dependencies

None

5.7.3 Race Condition Test for Attacker

Test Items

Attacker

Input Specification

None

Output Specification

None

Environmental Needs

A Pentium® III or higher machine.

Red Hat Linux (Family)

Special Procedural requirements

None

Intercase Dependencies

None

5.7.4 Race condition Test for Monitoring Agent

Test Items

Monitoring Agent

Input Specification

None

Output Specification

None

Environmental Needs

A Pentium® III or higher machine.

Red Hat Linux (Family)

Special Procedural requirements

None

Intercase Dependencies

None

5.7.5 Stress Test for Manager

Test Items

Manager

Input Specification

We simultaneously executed many disk intensive applications demanding resources in abnormal quantity, frequency, or volume.

Output Specification

None

Environmental Needs

A Pentium® III or higher machine.

Red Hat Linux (Family)

Special Procedural requirements

None

Intercase Dependencies

None

5.7.6 Performance Test for Manager

Test Items

Manager

Input Specification

None

Output Specification

None

Environmental Needs

A Pentium® III or higher machine.

Red Hat Linux (Family)

Special Procedural requirements

None

Intercase Dependencies

None

5.7.7 Performance Test for Route Tracing Agent

Test Items

Route Tracing Agent

Input Specification

None

Output Specification

None

Environmental Needs

A Pentium® III or higher machine.

Red Hat Linux (Family)

Special Procedural requirements

None

Intercase Dependencies

None

5.7.8 Performance Test for Information Gathering Agent

Test Items

Information Gathering Agent

Input Specification

None

Output Specification

None

Environmental Needs

A Pentium® III or higher machine.

Red Hat Linux (Family)

Special Procedural requirements

None

Intercase Dependencies

None

5.7.9 Performance Test for Monitoring Agent

Test Items

Monitoring Agent

Input Specification

None

Output Specification

None

Environmental Needs

A Pentium® III or higher machine.

Red Hat Linux (Family)

Special Procedural requirements

None

Intercase Dependencies

None

5.7.10 Class Test for IDRMAT

Test Items

Intrusion Detection and Response using Mobile Agent Technology

Input Specification

None

Output Specification

None

Environmental Needs

A Pentium® III or higher machine.

Red Hat Linux (Family)

Special Procedural requirements

None

Intercase Dependencies

None

5.7.11 Use Based Test for IDRMAT

Test Items

Intrusion Detection and Response using Mobile Agent Technology

Input Specification

None

Output Specification

None

Environmental Needs

A Pentium® III or higher machine.

Red Hat Linux (Family)

Special Procedural requirements

None

Intercase Dependencies

Class based test for IDRMAT

5.7.12 Regression Based Test for IDRMAT

Test Items

Intrusion Detection and Response using Mobile Agent Technology

Input Specification

None

Output Specification

None

Environmental Needs

A Pentium® III or higher machine.

Red Hat Linux (Family)

Special Procedural requirements

None

Intercase Dependencies

Use based test for IDRMAT

5.7.13 Cluster Test for IDRMAT

Test Items

Intrusion Detection and Response using Mobile Agent Technology

Input Specification

None

Output Specification

None

Environmental Needs

A Pentium® III or higher machine.

Red Hat Linux (Family)

Special Procedural requirements

None

Intercase Dependencies

Regression based test for IDRMAT

5.7.14 Behavior Test for IDRMAT

Test Items

Intrusion Detection and Response using Mobile Agent Technology

Input Specification

None

Output Specification

None

Environmental Needs

A Pentium® III or higher machine.

Red Hat Linux (Family)

Special Procedural requirements

None

Intercase Dependencies

Cluster based test for IDRMAT

5.7.15 Security Test for IDRMAT

Test Items

Intrusion Detection and Response using Mobile Agent Technology

Input Specification

None

Output Specification

None

Environmental Needs

A Pentium® III or higher machine.

Red Hat Linux (Family)

Special Procedural requirements

None

Intercase Dependencies

Behavior test for IDRMAT

5.7.16 User Interface Test for IDRMAT

Test Items

Intrusion Detection and Response using Mobile Agent Technology

Input Specification

None

Output Specification

None

Environmental Needs

A Pentium® III or higher machine.

Red Hat Linux (Family)

Special Procedural requirements

None

Intercase Dependencies

Security test for IDRMAT

5.7.17 Scenario Based Test for IDRMAT

Test Items

Intrusion Detection and Response using Mobile Agent Technology

Input Specification

None

Output Specification

None

Environmental Needs

A Pentium® III or higher machine.

Red Hat Linux (Family)

Special Procedural requirements

None

Intercase Dependencies

User interface test for IDRMAT

5.7.18 Random Test for IDRMAT

Test Items

Intrusion Detection and Response using Mobile Agent Technology

Input Specification

None

Output Specification

None

Environmental Needs

A Pentium® III or higher machine.

Red Hat Linux (Family)

Special Procedural requirements

None

Intercase Dependencies

Scenario based test for IDRMAT

Chapter 6

Conclusion

6 Conclusion

Mobile agent technology offers much to the field of intrusion detection and response. The idea of mobile and autonomous components intuitively seems useful in intrusion detection and response and many other applications. However, it is difficult to realize the benefits of mobile agent technology in practice. Despite these difficulties, the technology appears to provide valuable extensions to current capabilities. Although the barriers to creating practical mobile agent systems are high, the ability to move a running program from one hardware platform to another is a useful feature. Ultimately, as the security, performance, emerging technology, and standards barriers that inhibit this technology fall, mobile agents will enter mainstream use.

There are three main research areas for using MAs to do intrusion detection

1. Performance enhancements
2. Intrusion detection design improvements
3. Response improvements

Performance of an intrusion detection and response can be enhanced by creating light weight mobile agents because time is most critical constraint in security applications.

Within design improvements there are three categories of research

1. New detection paradigms
2. New architecture paradigms

Improvements over existing designs

Further, the area of responding the detected intruders at network level is nearly untouched yet. Work is required on this area also. To respond the detected attack, response system may be automated because there is no such advantage of detecting an attack without responding it in an automated way at computer speed so that damage may be minimized.

References and Bibliography

7 References and Bibliography

1. James P Anderson, "Computer Security Threat Monitoring and Surveillance," Technical Report, James P. Anderson Co., Fort Washington, PA, April 1980.
2. Dorothy E. Denning, "An Intrusion Detection Model," IEEE Transactions on Software Engineering, Vol. SE-13, No. 2, pp. 222-232, February 1987.
3. Stephen E. Smaha, "Haystack: An Intrusion Detection System, " Fourth Aerospace Computer Security Applications Conference, Orlando Florida, pp. 37-44, December 1988.
4. Teresa F. Lunt and R. Jagannathan, "A Prototype Real-Time Intrusion-Detection Expert System,"IEEE Symposium on Security and Privacy, April 1988.
5. Michael M. Sebring et ali, "Expert Systems in Intrusion Detection: A Case Study," National Computer Security Conference, pp. 74-81, October 1988.
6. L.Todd Heberlein, G.V. Dias, K. N. Levitt, B. Mukherjee, J. Wood, D. Wolber., "A Network Security Monitor," Procceedings of the Symposium on Research in Security and Privacy, pp. 296-304, May 1990.
7. Biswanath Mukherjee, L. Todd Heverlein, and Karl N. Levitt, "Network Intrusion Detection," IEEE Network, pp. 26-41, May/June 1994.
8. Frincke, D., Don Tobin, Jesse McConnell, Jamie Marconi, Dean Polla, "A Framework for Cooperative Intrusion Detection," National Information Systems Security Conference, pp. 361-373, October 1998.
9. Gregory B. White, Eric A. Fisch, and Udo W. Pooch. "Cooperating Security Managers: A peer-based intrusion detection system," IEEE Network, 10(1), pp. 20-23, January/February 1996.
10. W. Jansen, P. Mell, T. Karygiannis and D. Marks "Mobile agents in Intrusion Detection", proceedings of the 12th annual Canadian Information Technology Security Symposium, Ottawa, Canada, June 2000
11. "Lightweight Agents for Intrusion Detaction" By guy Helmer, Johnny S.K.Wong, Vasant Honavar,Les Miller Dapartment of computer Science Lowa State University November 2,2000

12. C. A. Carver, J. M. D. Hill, J. R. Surdu, and U. W. Pooch, "A Methodology for using Intelligent Agents to provide Automated Intrusion Response," in Proceedings of the IEEE Systems, Man, and Cybernetics Information Assurance and Security Workshop, West Point, NY, June 6-7, 2000.
13. Wayne Jansen, Tom Karygiannis, "Mobile Agent Security", Computer Security Division, NIST Special Publication 800-19,june 2001
14. Wayne A. Jansen, "Intrusion detection with mobile agents", National Institute of Standards and Technology, April,2001

Books

15. Software Engineering A Practitioner's Approach (Fourth Edition) by Roger S. Pressman 1992, McGraw-Hill Companies Inc.
16. Mobile Agents By William T. Cockayne and Michael zyda Manning Publishing Co.

Other

17. Biometric Security System Technology Page.

Research Paper

Second National Workshop on

Trends in Information Technology

(NWTIT 2003)

15 – 16 November 2003

Note:-Our paper is selected on 30th Number in the following list

Level 1 selection means the work will be presented in the NWTIT as an invited lecture. Its title and name of authors will be included in the book of proceedings but the paper will not be part of the publication.

Level 2 selection means the work will be presented in the NWTIT and the paper be published in the book of proceedings. Printing of the book will be delayed a little after the commencement of NWTIT in order to give time to the authors for improving the write-up. Meeting of the authors will be arranged with the paper referees during the workshop to discuss the problems and possible improvements in their papers.

Referees from PJIT will select the papers to be published in PJIT. We call this Level 3 selection.

Hence our objective is to improve the standard of research and provide training to the young researchers rather than keeping them away from the process by strictly rejecting the papers.

Sr. No.	Title of Paper	Author (s)	Result
30	Intrusion Detection and Response Using Mobile Agent Technology	Muhammad Ashraf Nadeem, Muhammad Sher	Selected for Level 2

Intrusion Detection and Response using Mobile Agent Technology

Muhammad Ashraf Nadeem
Department of Computer Science, International Islamic University, Islamabad
Anadeem_ch@yahoo.com

Muhammad Sher
Department of Computer Science, International Islamic University, Islamabad
Msher313@yahoo.com

Abstract

Today the computer security community is in search of novel solutions to achieve efficient detection and response mechanisms. It is especially because attackers intervene in an automated way, at computer speed. Therefore, there is need of such intrusion detection and response systems, which detect and respond at the same speed so that damage may be minimized.

We have designed an intrusion detection and response system prototype based on mobile agents. Our agents travel between systems in a network, obtain information, classify & correlate the information and report to the manager of the intrusion detection and response system (IDRS) which is responsible for responding the attack.

1. Introduction

The number of information warfare attacks is increasing day by day and are becoming increasingly sophisticated. Annual reports from the Computer Emergency Response Team (CERT) indicate a significant increase in the number of security incidents each year. Fig 1 shows the rise in computer security incidents. Only six incidents were reported in 1989 and over 8200 were reported in 1999[12].

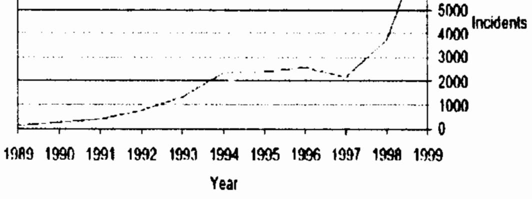


Figure 1: CERT Reported Incidents per Year

This paper offers a methodology of using mobile agents for intrusion detection and response and we propose an efficient solution for Intrusion detection and response. After providing some background information, we specify the problems found in current intrusion detection systems and propose a potential solution offered by mobile agents.

This research will develop mobile agent architecture for distributed detection and response to attacks in large- scale enterprise networks. It will exploit the flexibility of mobile agents to more efficiently detect and respond to intrusions by significantly reducing the time to do so because time is most critical constraint in computer security. The architecture to be developed is application architecture and is operating system dependent (i.e. Linux) and mobile agent system architecture (D's Agents). We also have a look at new approaches for automating response to an intrusion, once detected.

2. Background

The concept of intrusion detection was first proposed by James Anderson [1] in 1980, did not blossom until 1987 when Dorothy Denning published her seminal intrusion detection model [2]. Early IDS implementations employ a monolithic architecture whereby data collected at a single host was analyzed at a central point, at or adjacent to the point of collection [3, 4, 5]. Because monitoring account activity on a single host does not reveal attacks involving multiple hosts, IDS designers subsequently developed network-based IDSs that use a model of the network traffic to infer anomalies or misuses from low-level network packets traveling among hosts [6]. Network-based IDSs can be characterized as a change in perspective from host-centric to network-centric detection. A network-centric approach resolves a number of performance and integrity problems as well as problems associated with the reliance on audit trails. [7].

The US government provided significant funding for research in IDSs realizing that its computer systems were insecure. Hundred of millions of dollars have probably been spent on IDS research within last fifteen years [11].

Nearly all present-day commercial IDSs follow a hierarchical architecture. Information gathering occurs at leaf nodes, network-based or host-based collection points. Event information is passed to internal nodes that aggregate information from multiple leaf nodes. Further aggregation, abstraction, and data reduction can occur at higher internal nodes until the root node is reached. The root is a command and control system that evaluates attack situations and issues responses. The root typically reports to an operator console where an administrator can manually assess status and issue commands.

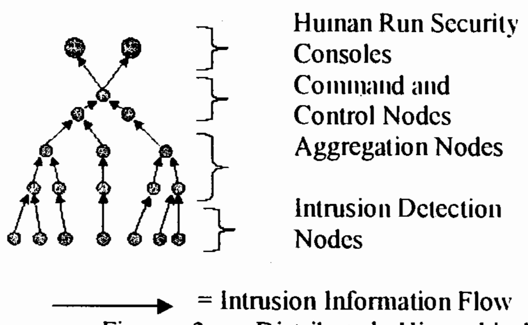

Human Run Security Consoles
Command and Control Nodes
Aggregation Nodes
Intrusion Detection Nodes
= Intrusion Information Flow

Figure 2: Distributed Hierarchical Intrusion Detection Architecture

In general, hierarchical structures result in efficient communications, whereby refined information filters upward in the hierarchy and control downward. The architecture is excellent for creating scalable distributed IDSs with central points of administration, but somewhat rigid because of the tight binding between functionality and lines of communication that tend to evolve. While IDS components tend implicitly toward a hierarchy, this tendency is not strict. Communications can occur, in general, between any type of components and not solely on a one-to-one or master/slave basis. For example, to improve notification and response, a collection unit may directly communicate a critical event to the command and control node, as well as an aggregation node. Moreover, peer relationships among command and control nodes are needed when different administrations manage portions of an enterprise network, or distinct and separate networks [8].

At least one IDS design, Cooperating Security Managers [9], uses a network structure, where information flows from any node to any other node, by consolidating the collection, aggregation, and command and control functions into a single component residing on every monitored system. Any significant events occurring at one system that stem from a connection originating from another are reported back to the system manager of the originating system by the security manager at the system where the event occurred. In situations where the originating system of the connection is an intermediate node in a communication chain, the system manager is obliged to report onward to the next system manager in the chain. Because of the potential for unconstrained communication flow, network structures, in general, tend to suffer from communications inefficiency when taken to the extreme (i.e., everyone directly communicating with everyone else). They can however, compensate for this inefficiency with flexibility in function.

3: Shortcomings of Current Intrusion Detection Systems

Present-day IDSs are less than perfect. Developers continue to address shortcomings through the improvement and refinement of existing techniques, but some shortcomings are inherent in the way IDSs are constructed [11, 12]. The most common shortcomings include the following:

1. Lack of Efficiency
2. High Number of False Positives
3. Burdensome Maintenance
4. Limited Flexibility
5. Vulnerability to Direct Attack

4. Advantages of Applying Mobile Agents

Mobile agent technology can potentially overcome a number of limitations intrinsic to existing IDSs that employ only static components. For example, mobility and autonomy make them ideal for detection schemes that follow a "cop on the beat," "immune system," or other real-world model.[10] A number of advantages of using mobile agent computing paradigms have been proposed and are relevant for intrusion detection systems[13, 14].

4.1. Overcoming Network Latency

Mobile agents can be dispatched to carry out operations directly at the remote point of interest, allowing them to respond in real time to changes in their environment. In addition to detecting and diagnosing potential network intrusions, mobile agents can provide appropriate response mechanisms. Such actions include gathering attack information sent to or emitted by the target of an attack, shutting down or isolating a system under attack to protect it from further damage, tracing the path of an attack, and shutting down or isolating an attacker's system.

4.2. Reducing Network Load

Instead of transferring the data across the network, mobile agents are dispatched to the machine on which the data resides, essentially moving the computation to the data, instead of moving the data to the computation, thus reducing the network load. A side benefit where confidentiality is a concern, is the efficiency of moving an encrypted agent and its refined data versus all of the raw data in encrypted form.

4.3. Autonomous and Asynchronous Execution

For large distributed systems the ability of the system to continue to operate when portions of it are destroyed or become isolated is essential. Mobile agents can exist and function independently from the creating platform, making them useful as IDRS components.

4.4. Dynamic Adaptation

The ability for mobile agent systems to sense their environment and react to changes is useful in intrusion detection. Agents may move elsewhere to gain better position or avoid danger, clone themselves for redundancy and parallelism, or marshal other agents for assistance. When combined with autonomous and asynchronous execution, these characteristics facilitate the building of robust and fault-tolerant systems.

Besides these advantages, mobile agents allow for a natural way to structure and design an IDRS. The agent orientation and mobility considerations provide an effective way for organizing data and functionality. Although our interest is in applying mobile agents to intrusion detection and response, it is unlikely that full mobility of all components would ever be effective in practice, due to the associated overhead. Therefore, some IDRS components may end up as static agents or remain static once deployed. Doing so allows the software agent paradigm to be applied and mobility to be used only where appropriate. Other practical factors such as trust relationships, performance capabilities, and location may also restrict mobile agents to a subset of available agent platforms.

5. Proposed System

In developing Intrusion detection and response system (IDRS), we propose a new intrusion detection and response model. IDRS will reduce the overhead of the system and detect new or unknown forms of attack. Our goal is not to detect all intrusions precisely but to detect many intrusions efficiently. To accomplish this goal, our system works by watching events that may relate to intrusions named as Marks Left by Suspected Intruder (MLSI).An MLSI is found when one of these events will occur.

1. Modification of critical files such as /etc/passwd, /etc/hosts.equiv, /etc/shadow and /.rhosts
2. su -id command was issued

Instead of analyzing all of the users' activities, if an MLSI is found, IDRS will gather information related to the MLSI, analyze the information, and decide whether an intrusion has occurred. For example, IDRS monitors whether or not critical files related to system security have been modified, since, in many cases, intruders tamper with them. However, because legitimate users 'may also change the files, the system cannot conclude solely based on file modifications that an intrusion has occurred. IDRS therefore gathers further information related to the modification of the file before deciding if an intrusion has occurred.

6. Structure of IDRS

In many present day conventional network intrusion detection systems, each target system transfers its system log to an intrusion-detection server, and the server analyzes the entire log in search of intrusions. In a large-scale network deploying an intrusion detection system, network traffic will be extremely high, since the volume of the system logs that are routinely transferred is very large, though most of it has no information related to intrusions. Therefore, this type of intrusion detection system on a large-scale network does not fulfill its function efficiently. To solve this problem, we adopted a mobile-agent paradigm in developing IDRS. Mobile agents autonomously migrate to target systems to collect only information related to intrusions, eliminating the need to transfer system logs to the server. IDRS consists of a manager, monitoring agent, manager board, information register, route-tracing agents, and information-gathering agents.

6.1. Monitoring Agent

The monitoring agent, present on each host, monitors system logs in search of MLSIs. If a monitoring agent finds an MLSI, it reports this finding to the manager. The monitoring agent also reports on the type of MLSI.

6.2. Route Tracing Agent (RTA)

The intrusion-route tracing agent traces the path of an intrusion and identifies its point of origin, the place from which the user leaving an MLSI remotely logged onto the target host. In the course of finding the origin, a tracing agent can find any intermediate nodes that were compromised. When a RTA goes to the next system, first it checks information register. If there is no information in the information register about this particular MLSI, the RTA then completes its task, enters information in the information register and moves on to the next compromised system. If information related to that MLSI already exists in the information register meaning that another agent has already traced for this particular MLSI. Then the tracing agent enters its reference in the information register and returns to the manager. At a specific point, where RTA cannot proceed to the next system, it means that this particular system is the origin of the intrusion. So RTA returns to the manager after putting information into the information register.

6.3. Information Gathering Agent (IGA)

An information-gathering agent collects information related to MLSIs from a target system. Each time an RTA in tracking down of an intruder is dispatched into a target system by the manager, it activates an information-gathering agent in that system. Then the information-gathering agent collects information depending on the type of MLSI, returns to the manager, and reports.

If the RTA migrates to another target system, it will activate another information-gathering agent on that system, which will gather information on that system. Many information-gathering agents may be activated by many different RTAs on the same target system. An information-gathering agent is not capable of deciding whether an intrusion has occurred or not. An Information gathering agent posts the information on the manager board. Depending on that information manager decides whether an intrusion has really occurred or not. IGA puts the following information on the manager board.

1. ID of the information gathering agent.
2. The name of the target system from where information was gathered.
3. The name of the target system preceding the target where information was gathered.
4. Information gathered on the target system

6.4. Manager

The manager analyzes information gathered by information-gathering agents and detects intrusions. It manages Manager Board and process of dispatching the RTAs and provides an interface between administrators and the system. The manager accumulates and weighs the information entered by the IGAs on the manager board, and if the weights exceed a set threshold, the manager concludes that an intrusion has occurred. After dispatching RTAs, the manager has no concern with the movement of RTAs. i.e. where the RTA will go after a specific system.

6.5. Manager Board

This is on the manager's machine, which is a means of exchanging information among IGAs because it is shared area, accessible by all IGAs. Information gathering agents (IGAs) put tracing agents to both targets D and B. The tracing agent DA traces intrusions in the following order: D - C - B - A. Tracing agent BA, on the other hand, traces in the following order: B - A. The two agents' tracings therefore overlap on B – A

9. Response Mechanisms

Our main emphasis was to detect intruders but we gave a light touch to response to intruders. However, humans cannot automate what they themselves cannot do. For responding the detected intruders, we present following mechanisms.

1. Response at the host level
2. Response at the network level

9.1. Response at the Host Level

Responding at the host level, the following two mechanisms may be applied

1. Responding at the target
2. Responding at the source

9.2. Responding at the Target

After detecting an attack, it is essential to automatically respond at the target host. A quick response can prevent the attacker from establishing a better foothold and using the penetrated host to further compromise the network. It can also minimize the effort needed to recover damage done by the attacker.

9.3. Responding at the Source

Responding at the attacker's host gives IDRS a much greater power to restrict the attacker's actions. Without using mobile agents, it is unlikely that an IDRS would have sufficient access to an attacker's host in order to take corrective action. While this option has limitations, since it requires an agent platform be active on the attacker's host and the attack to come from within the management domain, it also has the potential to be a very effective part of the IDRS.

At network level, there are further three mechanisms

1. Isolating the target
2. Isolating the source
3. Kill TCP connection between source and target.

10. Conclusion and Future Work

Mobile agent technology offers much to the field of intrusion detection and response. The idea of mobile and autonomous components intuitively seems useful in intrusion detection and response and many other applications. However, it is difficult to realize the benefits of mobile agent technology in practice. Despite these difficulties, the technology appears to provide valuable extensions to current capabilities. Although the barriers to creating practical mobile agent systems are high, the ability to move a running program from one hardware platform to another is a useful feature. Ultimately, as the security, performance, emerging technology, and standards barriers that inhibit this technology fall, mobile agents will enter mainstream use.

There are three main research areas for using MAs to do intrusion detection

1. Performance enhancements
2. Intrusion detection design improvements
3. Response improvements

Performance of an intrusion detection and response can be enhanced by creating light weight mobile agents because time is most critical constraint in security applications.

Within design improvements there are three categories of research

1. New detection paradigms
2. New architecture paradigms

Improvements over existing designs

Further, the area of responding the detected intruders at network level is nearly untouched yet. Work is required on this area also. To respond the detected attack, response system may be automated because there is no such advantage of detecting an attack without responding it in an automated way at computer speed so that damage may be minimized.

11. Acknowledgments

Special thank to Dr. Muhammad Saleem Rathore (PAEC) and Dr.Hafiz Farooq Ahmed (NIIT, NUST) who gave us their precious time and helped us a lot to figure out the problems and to get rid of confusions.

Furthermore ,many thanks to Muneeb Ahmed Awan and S.Tauseef-ur-Rehman for their co-operation.

12. References

[1] James P Anderson. "Computer Security Threat Monitoring and Surveillance." Technical Report. James P. Anderson Co., Fort Washington, PA. April 1980.

[2] Dorothy E. Denning. "An Intrusion Detection Model." IEEE Transactions on Software Engineering, Vol. SE-13, No. 2, pp. 222-232. February 1987.

[3] Stephen E. Smaha. "Haystack: An Intrusion Detection System. " Fourth Aerospace Computer Security Applications Conference. Orlando Florida. pp. 37-44. December 1988.

[4] Teresa F. Lunt and R. Jagannathan. "A Prototype Real-Time Intrusion-Detection Expert System."IEEE Symposium on Security and Privacy. April 1988.

[5] Michael M. Sebring et ali. "Expert Systems in Intrusion Detection: A Case Study." National Computer Security Conference, pp. 74-81. October 1988.

[6] L.Todd Heberlein, G.V. Dias, K. N. Levitt. B. Mukherjee. J. Wood, D. Wolber.. "A Network Security Monitor," Proccccdings of the Symposium on Research in Security and Privacy, pp. 296-304, May 1990.

[7] Biswanath Mukherjee, L. Todd Heverlein, and Karl N. Levitt, "Network Intrusion Detection." IEEE Network, pp. 26-41, May/June 1994.

[8] Frincke, D., Don Tobin, Jesse McConnell, Jamie Marconi, Dean Polla, "A Framework for Cooperative Intrusion Detection," National Information Systems Security Conference, pp. 361-373, October 1998.

[9] Gregory B. White, Eric A. Fisch, and Udo W. Pooch. "Cooperating Security Managers: A peer-based intrusion detection system." IEEE Network, 10(1), pp. 20-23, January/February 1996.

[10] W. Jansen, P. Mell, T. Karygiannis and D. Marks "Mobile agents in Intrusion Detection". proceedings of the 12th annual Canadian Information Technology Security Symposium, Ottawa, Canada, June 2000

[11] "Lightweight Agents for Intrusion Detaction" By guy Helmer, Johnny S.K.Wong. Vasant Honavar,Les Miller Dapartment of computer Science Lowa State University November 2.2000

[12] C. A. Carver, J. M. D. Hill, J. R. Surdu, and U. W. Pooch. "A Methodology for using Intelligent Agents to provide Automated Intrusion Response." in Proceedings of the IEEE Systems, Man, and Cybernetics Information Assurance and Security Workshop, West Point, NY, June 6-7, 2000.

[13] Wayne Jansen, Tom Karygiannis, "Mobile Agent Security". Computer Security Division. NIST Special Publication 800-19,june 2001

[14] Wayne A. Jansen, "Intrusion detection with mobile agents". National Institute of Standards and Technology. April.2001