AN AGENT BASED DISTRIBUTED INTRUSION DETECTION SYSTEM

A DISSERTATION

Submitted in partial fulfillment of the requirements for the award of the degree

of

MASTER OF TECHNOLOGY

in

INFORMATION TECHNOLOGY

By

R. SOWMYA

DEPARTMENT OF ELECTRONICS AND COMPUTER ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY ROORKEE
ROORKEE-247 667 (INDIA)

JUNE, 2009

Candidate's Declaration

I hereby declare that the work being presented in the dissertation report titled "AN AGENT BASED DISTRIBUTED INTRUSION DETECTION SYSTEM" in partial fulfillment of the requirement for the award of the degree of Master of Technology in Information Technology, submitted in the Department of Electronics and Computer Engineering, Indian Institute of Technology Roorkee, is an authenticate record of my own work carried out under the guidance of Dr. Kumkum Garg, Professor, Department of Electronics and Computer Engineering, Indian Institute of Technology Roorkee.

I have not submitted the matter embodied in this dissertation report for the award of any other degree.

Dated: 30-06-09

Place: IIT Roorkee.

[illegible]

(R. Sowmya)

Certificate

This is to certify that above statements made by the candidate are correct to the best of my knowledge and belief.

Dated: 30-06-09

Place: IIT Roorkee.

[illegible]

Dr. Kumkum Garg,
Professor,
Department of Electronics and
Computer Engineering, IIT Roorkee,
Roorkee - 247667 (India).

ACKNOWLEDGEMENTS

I am thankful to Indian Institute of Technology Roorkee for giving me this opportunity. It is my privilege to express thanks and my profound gratitude to my supervisor Prof. Kumkum Garg for her invaluable guidance and constant encouragement throughout the dissertation. I was able to complete this dissertation in this time due to constant motivation and support obtained from Prof. Kumkum Garg.

I am also grateful to the staff of Network security laboratory for their kind cooperation extended by them in the execution of this dissertation. I am also thankful to all my friends who helped me directly and indirectly in completing this dissertation.

Most importantly, I would like to extend my deepest appreciation to my family for their love, encouragement and moral support. Finally I thank God for being kind to me and driving me through this journey.

(R. SOWMYA)

ABSTRACT

An Intrusion Detection System (IDS) is an automated system that aims to detect intrusions or attacks in a computer system. The main goal of IDS is to detect any unauthorized use, abuse, or misuse of computer system by both system insiders and external attackers. The IDS architectures commonly used Centralized IDS, but these systems suffer from single point of failure and at heavy load these CIDS may not detect all attacks. That limits their configurability, scalability and efficiency. The difficulty of these IDS leads the idea of agents based IDS.

In this work, a novel IDS is proposed which addresses the problems of existing Centralized IDS. This proposed system uses agents along with a Network Intrusion Detection System (NIDS) to efficiently detect and trace back an internal attacker. The proposed system satisfies all necessary requirements, i.e. it should be easily and frequently updated with new attack signatures, it should adapt to changes in network topology and it should detect anomalous events or beaches in security should be detected in real-time and reported immediately.

To eliminate single point of failure in the system proposed, NIDS are replicated at the secondary monitor. Existing Distributed Intrusion Detection Systems send whole system log, thus requiring a larger bandwidth, but in system proposed Agents send only required results to the monitor station, thus requiring a smaller bandwidth. The system uses misuse detection model for detecting attacks in the network.

The proposed architecture has been developed in Java. This system uses IBM Aglet 2.0.12 to provide a mobile agent environment, the open source database-Mysql as the background DB , gcc 4.3.1 for generating attack, inotify-java, which is a Linux kernel subsystem for file system event notification and open source jpcap 0.7 at monitor station for sniffing network data.

CONTENTS

1. INTRODUCTION

1.1 Introduction

In the last few years, there has been a tremendous increase in connectivity between systems which has brought about limitless possibilities and opportunities. Unfortunately, security related problems also increased at the same rate. Computer systems are becoming increasingly vulnerable to attacks. These attacks, based on flaws in operating systems or application programs, usually read or modify confidential information or render the system useless. Formally, an intrusion is defined as any activity that violates the confidentiality, integrity or availability of the system.

Intrusion prevention is more desirable, but it cannot be fully achieved due to several reasons like bugs in software, abuse by insider and human negligence. Many times it is difficult to have good access control while simultaneously making system user friendly. Attacks are inevitable, but even after the attack has occurred, it is important to determine that the attack has happened, assess the extent of damage and track down the attacker [1]. This helps in preventing further attacks. Due to these reasons, a detection system as a second line of defence is always desirable.

The concept of Intrusion Detection (ID) was first introduced by Anderson [2] in 1980, which was later refined by Denning [3] in 1987. Anderson defined intrusion as an intentional unauthorized attempt to access information, modify information, or make a system untrustworthy or unusable. Hence, ID is the process of detecting unauthorized access to the system which violates confidentiality, integrity or availability policies of the system.

The main goal of Intrusion Detection System (IDS) is to analyze events on the network and identify attacks. . The IDS architectures commonly used Centralized IDS, but these systems suffer from single point of failure and at heavy load these CIDS may not detect all attacks. This limits their configurability, scalability and efficiency. The difficulty of these IDS leads to the idea of agents. Detecting intrusion in distributed networks, from outside the network as well as from inside the network is very difficult. An IDS has to analyze the large volume of data while not placing load on monitoring system's network. Mobile agent technology can provide IDS flexibility and enhanced distributed detection ability. Agents can detect and take predefined actions against malicious activity. The Distributed Intrusion System shows a superior performance compared to existing monolithic IDS techniques. This is one of the major motivations to use the distributed model based on Mobile Agent platform.

1.2 Problem Statement

To design an efficient agent based Distributed Intrusion Detection System.

1.3 Organization of Report

Including this introduction chapter, this report contains 6 chapters.

Chapter 2 presents an overview of Intrusion Detection Systems and mobile agents. The several characteristics that are desirable for IDS have also been discussed. The advantages of using Agents in IDS are discussed.

Chapter 3 gives a review of existing Distributed Intrusion Detection System. The advantages and disadvantages of existing techniques are mentioned.

Chapter 4 presents the architecture and essential components of the proposed system with the working of its essential components.

Chapter 5 gives implementation details and discusses the results.

Finally, the work is concluded and the future scope is given in chapter 6.

2. INTRUSION DETECTION SYSTEMS

2.1. Introduction

An intrusion is an event, or a set of events, that attempts to compromise a computer system's confidentiality, integrity, availability, or that attempts to bypass its security mechanisms. Intrusions can be caused by system insiders or by external attackers [4]. System insiders are users authorized to use the system, but they can cause intrusions by attempting to gain privileges to which they are not entitled or by misusing the privileges that have been given to them. External attackers are users who have not been authorized to use the system, and can cause intrusions by gaining access to the system from outside such as the internet.

An IDS collects and analyze data from different system and network resources to check any security breach; and alerts the system administrator on finding an intrusion. IDS raises an alert in case an outside intruder breaks into the system or an inside user escalates their privilege or misuses resources. It works just like a burglar alarm that raises an alarm in case of misuse.

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

There are several reasons why IDS are necessary [1]:

- To detect attacks and other security violations that other security measures cannot prevent. IDS can be used to detect attacks that exploit vulnerabilities in the security mechanisms of a computer system. In addition IDS can serve an important function in protecting the system, because it can report intrusions to system administrators, who can recover any resulting damage.
- The first stage of an attack usually involves examining a system for any vulnerability, searching for an optimal point of entry. This stage is often

experienced as network probes and other tests for existing vulnerabilities. By using an IDS probes can be detected and actions may be taken to block the attacker access to the target system.

- To act as a means of quality control for security design and administration. An IDS runs over a period of time can show patterns of system usage and detected problems. These can show the design and management of flaws in the system security. Deficiencies can be corrected before they cause a security problem.
- To provide information about actual intrusions. IDS can collect relevant and detailed information about the attack. This supports incident handling and recovery efforts. Such information can also be used to identify problem areas in the security configurations or policy of the system.

2.2. Classification of IDS

IDS are classified according to what analysis technique is used to detect intrusions. An ID mainly uses two techniques: misuse detection and anomaly detection [5].

2.2.1 Misuse Detection

Misuse Detection, also known as Signature based detection, identifies intrusion by matching a pattern of activities corresponding to a known intrusion in signature database. Misuse Detection only detects known or a small variation of known intrusions, because the signature of the intrusion must already be defined in signature database. It also provides intrusion detection confidence by producing a low rate of false positive (IDS raises an alert whereas no intrusion occurred in reality) due to detection of only known intrusions.

The size of signature database grows as new attacks are discovered. This causes a problem as new attacks emerge at an alarming rate. Secondly, to detect all known attacks signatures the database should be up-to-date. This creates a challenging task for the IDS administrator. Detection of only the known attacks makes it prone to novel attacks which can pass without any notice. Attackers can even launch the older attacks by slightly changing the attacking method to circumvent the signature.

2.2.2 Anomaly Detection

In Anomaly Detection a baseline of normal behaviour is initially established and any deviation from the normal behaviour is flagged as anomalous which results in triggering an alert. A baseline for normal behaviour (e.g. network load, application resource usage, packet size, etc.) is established either by the network administrator or through self learning. A threshold of the accepted deviation is also defined and any activity outside this threshold will be considered as an expected intrusion.

Anomaly detection can also detect unknown attacks deviating from the normal system usage unlike misuse detection that heavily relies on prerequisite knowledge of signature. Anomaly Detection IDSs are based on heuristics instead of their relying on signature database in case of misuse detection.

It is difficult to train anomaly based IDSs to learn every aspect of the normal traffic and when they fail to learn about the normal traffic, they trigger alarms by flagging the normal usage as anomalous. This induces the IDS to produce a high rate of false positives. This is a major challenge in case of anomaly detection IDSs.

An IDS requires specific type of data that it can analyse for possible intrusions. The data is obtained from different sources, depending on what types of attacks should be detected by the IDS. With respect to the source of data used for analysis, intrusion detection systems are classified as host-based or network based [1].

2.2.3 Host-based IDS

Host-based Intrusion Detection Systems (HIDS) are designed to monitor and detect attacks targeted to single host only. This can determine exactly which processes and users are involved in a particular attack on the operating system. A host based IDS have the ability to directly access and monitor the data files and system processes usually targeted by attacks. Therefore, it can view the system after an attempted attack, which allows it to verify the success of failure of an attack.

A host-based IDS normally uses information sources of two types: OS audit logs and system logs. Hence, HIDS produces a low rate of false positives. Operating system audit logs are records of system events, generated at the innermost (kernel) level of operating system. System logs, on the other hand, are files of system and application events.

The capability of HIDS to combat the internal threats limits their view to detect attacks targeted to more than one host like distributed and network attacks. These distributed attacks leave only the innocent marks on each system, which cannot be detected in case data is collected from a single system only. Moreover the size and diversity of networks makes it infeasible to have HIDS to detect attacks on each system in the network.

2.2.4 Network-based IDS

A Network-based Intrusion Detection System (NIDS) monitors traffic on the entire network segment in order to trace the malicious activities on the network. NIDS are put at strategic points in the network infrastructure by setting the network interface into promiscuous mode in order to scan the traffic destined also to the other hosts. Unlike HIDS, the dedicated machines can be used for NIDS which make them less prone to compromise by splitting and enforcing more security on them. NIDS can analyze the captured data by detecting known attacks by comparing it with the signature database.

Although NIDS can correlate data destined to different hosts, it requires efficient packet scanning, to meet ever increasing network size and bandwidth, to ensure that no threats are missed. This can result in scalability problems in case of high network load. NIDS can also produce a higher rate of false positives as compared to HIDS, because data is collected from diverse nodes. The best approach is to use a hybrid IDS which combines the best of both HIDS and NIDS in order to complement better false positive accuracy with a diverse range of detection capability.

A comparison of HIDS and NIDS is given in Table 2.1

Host-based IDS	**Network-based IDS**
Monitors the activities on a single host for malicious use.	Monitors the traffic on the entire network segment to trace the malicious activities.

HIDS are resource hungry and must reside on each host that needs detection.	Dedicated machines can be use for NIDS
HIDS is suitable for attacks originating from inside the network perimeter.	NIDS is suitable to detect attacks from outside the network.
It produces a low rate of false positives.	Comparatively higher rate of false positives
Should be OS dependent.	Can be OS independent.
Detect attacks targeted to single host only.	Detect attack at network level, targeted to multiple hosts.
Difficult to maintain HIDS in large networks.	Provides good infrastructure for large but have scalability limitations in very large networks.

Table 2.1 Host based IDS vs. Network based IDS

2.3. IDS Requirements

There are several characteristics that are desirable in IDS. Jansen et al. (1999) [3] have divided these characteristics into two groups of requirements: functional requirements and performance requirements.

2.3.1 Functional requirements

- The IDS must continually monitor and report intrusions.
- When an intrusion occurs, the IDS must supply enough information to determine the extent of the damage.
- The IDS should be easily and frequently updated with new attack signatures as new security advisories and security patches become available and as new vulnerabilities and attacks are discovered.
- The IDS should adapt to changes in network topology and configuration as computer devices are dynamically added and removed from the network.

2.4.1 Performance requirements

- To the extent possible, any anomalous events or breaches in security should be detected in real-time and reported immediately. This may minimize the damage to the network and the loss or corruption of data.
- The IDS should not impose a large overhead on the computer system.
- The IDS should be scalable to enable it to handle additional computational and communication load, as new computer devices are added to the network.

Our proposed IDS meet all the above requirements.

2.5. Limitations of existing IDS[6]

- **Lack of efficiency:** Current IDSs are not efficient enough to evaluate events in real-time with large number of events and on high-speed networks with large volumes of traffic.
- **High number of false positives:** Current IDSs have a high false positive rate because recognition of intrusions is not perfect.
- **Limited flexibility:** IDSs have typically been written for a specific environment and have proved difficult to use in other environments that may have similar policies and concerns.

2.5 Software Agents

As per the IBM's definition [7], an agent is a software object that is situated within an execution environment and acts on behalf of others in an autonomous fashion and exhibits some levels of the key attributes of learning, coöperation, and mobility.

2.5.1 Mobile vs. Stationary Agents

Mobility is an orthogonal property of agents. That is, all agents are not necessarily required to be mobile. An agent can remain stationary and communicate with the surroundings by conventional means like remote procedure calls (RPC) and remote object invocation (RMI). The agents that do not or cannot move are called stationary agents.

On the other side, a mobile agent is not bound to the system where it begins execution. The mobile agent is free to travel among the hosts in the network. Once created in one execution environment, it can transport its state and code with it to another execution environment in the network, where it resumes execution.

2.5.2 Mobile Agents and Mobile Agent Environment

A mobile agent must contain all of the following models:

1. Agent model
2. Life-cycle model
3. Computational model
4. Security model
5. Configuration model
6. Navigation model

Mobile agent consists of a self-contained piece of software that can migrate and execute on different machines in a dynamic networked environment, and that senses and (re) acts autonomously and proactively in this an environment to realize set of goals or tasks [8].

The software environment in which the mobile agents exist is called mobile agent environment. Mobile agent environment is a software system distributed over a network of heterogeneous computers. Its primary task is to provide an environment in which mobile agents can execute. It implements the majority of the models possessed by a mobile agent.

The mobile agent environment is built on top of a host system. Mobile agents travel between mobile agent environments. They can communicate with each other either locally or remotely.

2.5.3 Mobile Agent paradigm vs. Client-Server paradigm[9]

Client-server paradigm enjoys various techniques like remote procedure calling (RPC), remote object-method invocation (like Java RMI or CORBA) etc. The RPC paradigm, for example, is the prominent technique of the client-server paradigm. It views computer-to-computer communication as enabling one computer to call procedures in another. Each message that the network transports either requests or acknowledges a procedure's performance. Two computers whose communication follows the RPC paradigm have to agree upon the effects of each remotely accessible procedure and the types of its arguments and results. This agreement constitutes a protocol.

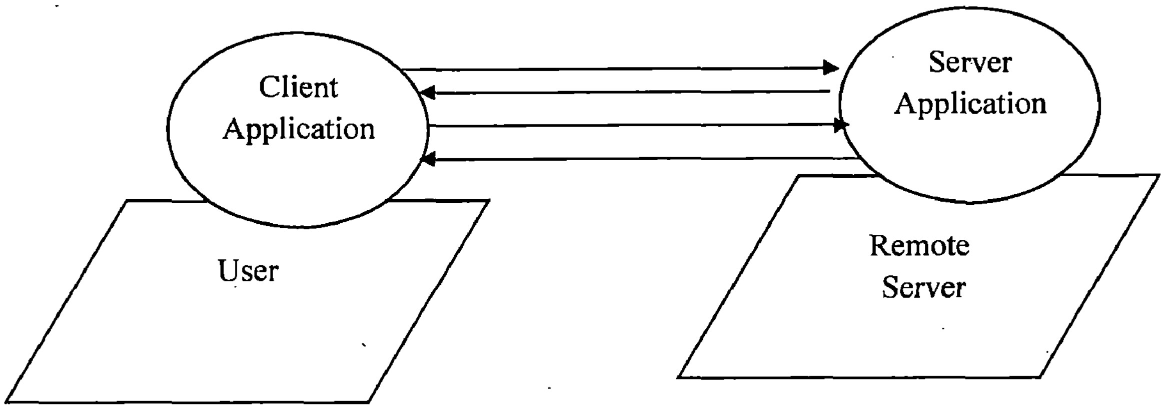


Figure2.1 Client Server Communication Paradigm

For an example, as shown in Figure 2.1, a client computer initiates a series of remote procedure calls with a server in order to accomplish a task. Each call involves a request sent from client to server and a response sent from server to client. Thus the salient feature of client-server paradigm is that each interaction between the client and the server requires two acts of communication. That is, ongoing interaction requires ongoing communication.

In contrast to client-server paradigm, the mobile agent paradigm views computer-to-computer communication as enabling one computer not only to call procedures in another, but also to supply the procedures to be performed. Each message that the network transports consists of a procedure. Two computers whose communication follows the mobile agent paradigm have to agree upon the instructions that are allowed in a procedure and the types of data that are allowed in its state. This agreement constitutes a language.

Figure 2.2 represents the same example scenario as before but using mobile agent paradigm. Here the client computer sends an agent to the server whose procedure there makes the required requests to the server. The dotted line in Figure 2.2 shows the previous movement of the agent. All the request and responses in this case are local to the server and no network is required to complete a task. Thus the salient feature of mobile agent paradigm is that each a client computer and a server can interact without using the network once the network has transported an agent between them.

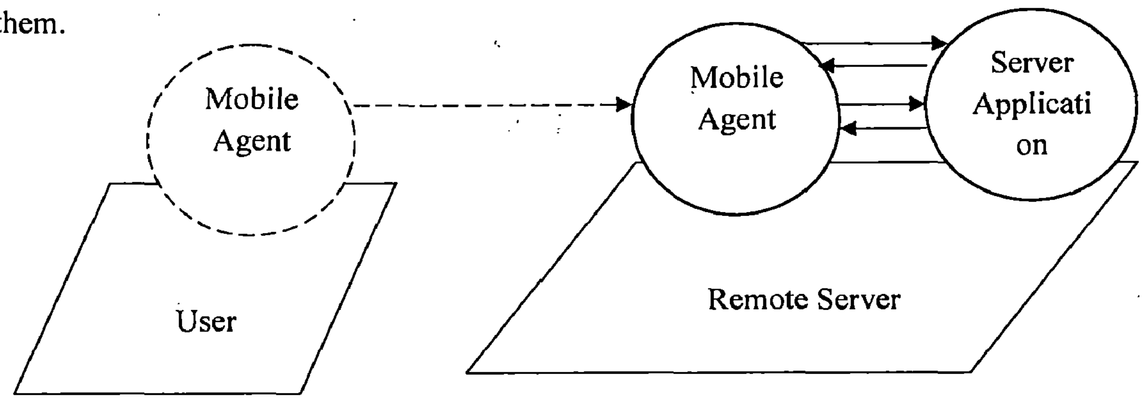


Figure2.2. Communications using mobile agent paradigm

The mobile agents have several strengths. The following is the brief discussion of five good reasons for using mobile agents [10]:

- **They reduce network load:** The main motivation behind using mobile agents is to move the communication to the data rather than the data to the computations. Distributed systems often required multiple interactions to complete a task. But using mobile agent allows us to package a conversation and send it to a destination host. Thus all the interactions can now take place locally. The result is enormous reduction of network traffic. Similarly instead of transferring large amount of data from the remote host and then processing it at the receiving host, an agent send to the remote host can processed the data in its locality.

- **They are naturally heterogeneous:** Mobile agents are generally independent of the computer and the transport layer and depend only on their execution environment. Hence they can perform efficiently in any type of heterogeneous networks.

- **They are robust and fault-tolerant:** The dynamic reactivity of mobile agents to unfavourable situations makes it easier to build robust and fault-tolerant distributed systems.

- **Overcoming network latency:** Network latency can be reduced by sending an agent with a sequence of service requests across the network rather than by issuing each service request by a separate remote procedure call.
- **Dynamic adaptation:** Mobile agents have the ability to sense their execution environment and autonomously react to changes.

2.5.4 Mobile Agent Frameworks (MAFs)

Many research/commercial MAFs have been developed and major review can be found in [11]. Java has been most popular with MAF developers, because of its platform independent, object oriented language construct, object serialization/de-serialization (suitable for migration) etc. We describe the Aglet framework which we have used.

2.5.5 Aglets

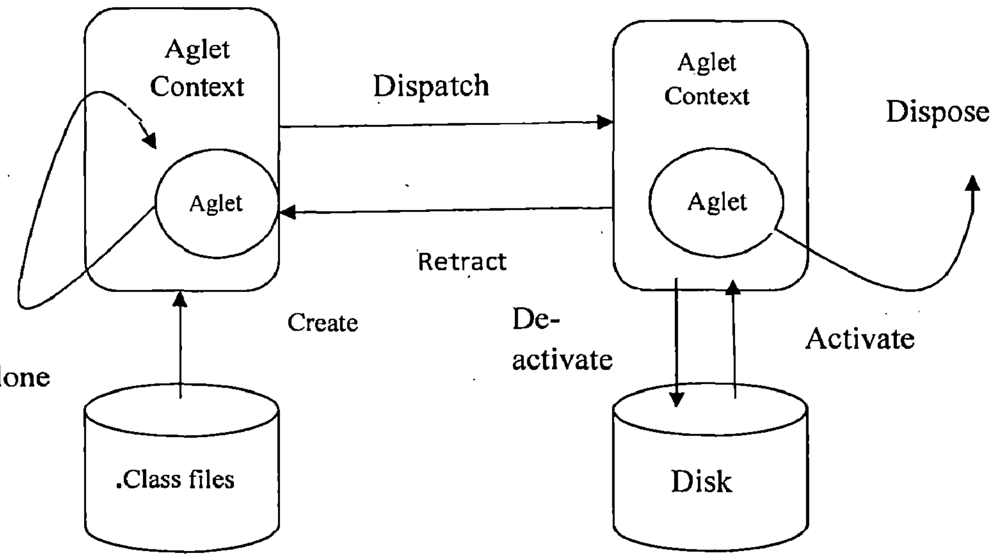


Figure 2.3 Aglets Life cycle Events

Aglet is defined as a mobile java object that visits Aglet enabled host in a computer Network. Aglets Software Development Kit [ASDK] is a product of IBM's Tokyo Research laboratory, initiated in early 1995. The goal has been to bring the flavour of mobility to Applets (Aglets means Agent plus Applet). The Aglets SDK includes Aglets API documentation, sample Aglets, the aglet server (TAHITI) and the agent web launcher (FIJI). Various Aglets abstractions, life cycle events defined by this SDK are shown in Figure 2.3.

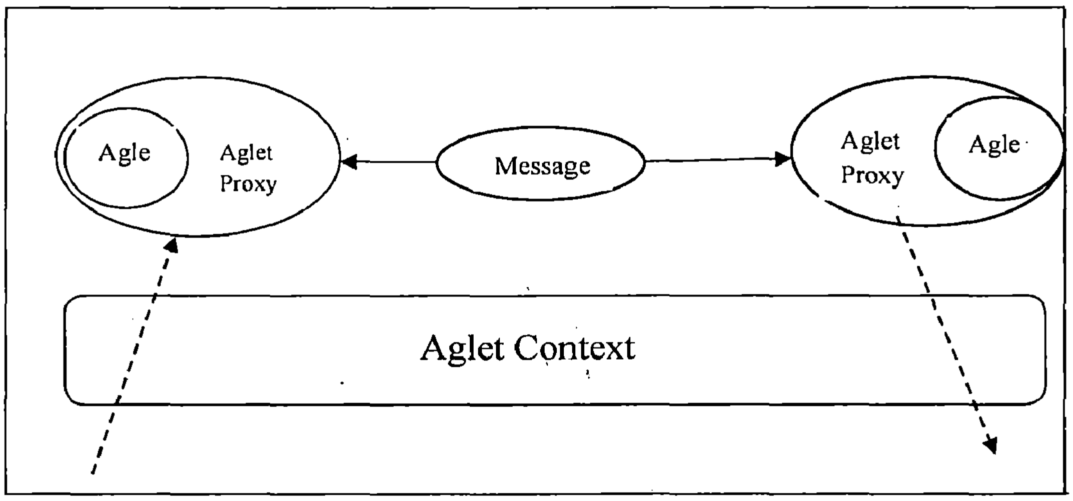


Figure 2.4 Aglet Context

Figure2.4 shows the Aglet Context. It comprises of the following:

- **Aglet** - Mobile Java object that runs in its own thread, acts autonomously, visits local and remote hosts, and reacts to events and messages.
- **Proxy** - Provides Aglet with location transparency and a shield from direct access.
- **Context** - Stationary workplace that hosts Aglets. Provides platform resources
- **Identifier** - Globally unique, immutable Aglet identifier. AgletID maintained in an AgletInfo object associated with Aglet.

3.1 Intrusion Detection Using Autonomous Agents (AAFID)

AAFID implements a host based hierarchical design [13]. Essential components of the architecture are agents, transceivers and monitors.

Each host can contain any number of agents that monitor for interesting events occurring in the host. All the agents in a host report their findings to a single transceiver. Transceivers are per-host entities that oversee the operation of all the agents running in their host. They may also perform data reduction on the data received from the agents. Finally, the transceivers report their results to one or more monitors. Monitors have access to network-wide data; therefore they are able to perform higher-level correlation and detect intrusions that involve several hosts. Also, a transceiver may report to more than one monitor to provide redundancy and resistance to the failure of one of the monitors. Figure 3.1 shows architecture of AAFID.

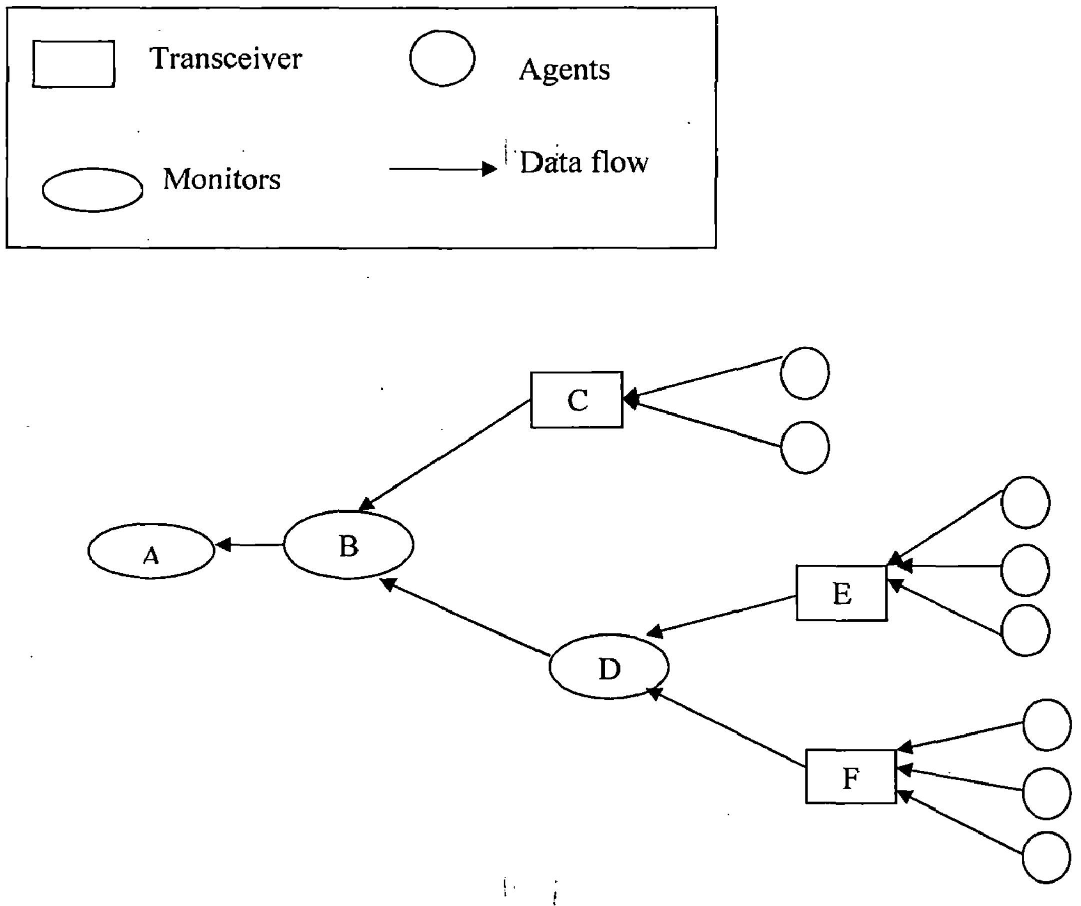


Figure3.1. Architecture of AAFID

3.2 Intrusion Detection Agent System (IDA)

IDA [14] consists of managers, sensors, bulletin boards, Message boards, Tracing agents, and information gathering agents. Figure 3.2 shows architecture of IDA.

Sensors: present on each target system, monitor system logs in search of MLSIs. If a sensor finds an MLSI, it reports this finding to the manager. The sensor also reports on the type of MLSI.

Manager: The manager analyzes information gathered by information-gathering agents (which are described below) and detects intrusions.

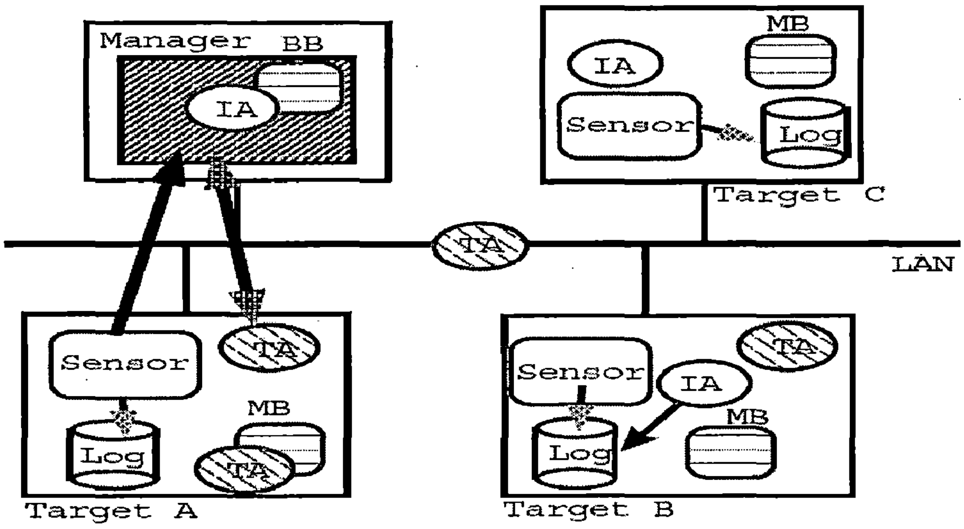


Figure3.2 IDA Architecture

TA: Tracing Agent
IA: Information gathering Agent
BB: Bulletin Board
MB: Message Board

Tracing agent: The intrusion-route tracing agent, called simply the 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.

Information-gathering agent: An information-gathering agent, which is mobile, gleans information related to MLSIs from a target system.

3.3 Intelligent Mobile Agent for Intrusion Detection System (IMAIDS)

IMAIDS [14] consists of following components. Figure 3.3 shows architecture of IMA-IDS.

Collector agent: This kind of agent will be cloned and distributed throughout the network. This agent patrols the network and collects all the events occuring in the host to which it is related.

Correlator agent: This creates contexts of connections. The contexts of connections represent the relations between various information coming from multiple distinct collector agents. This correlator agent uses a set of rules to classify crucial events and will hurry this specific information to the appropriate analyzer agent.

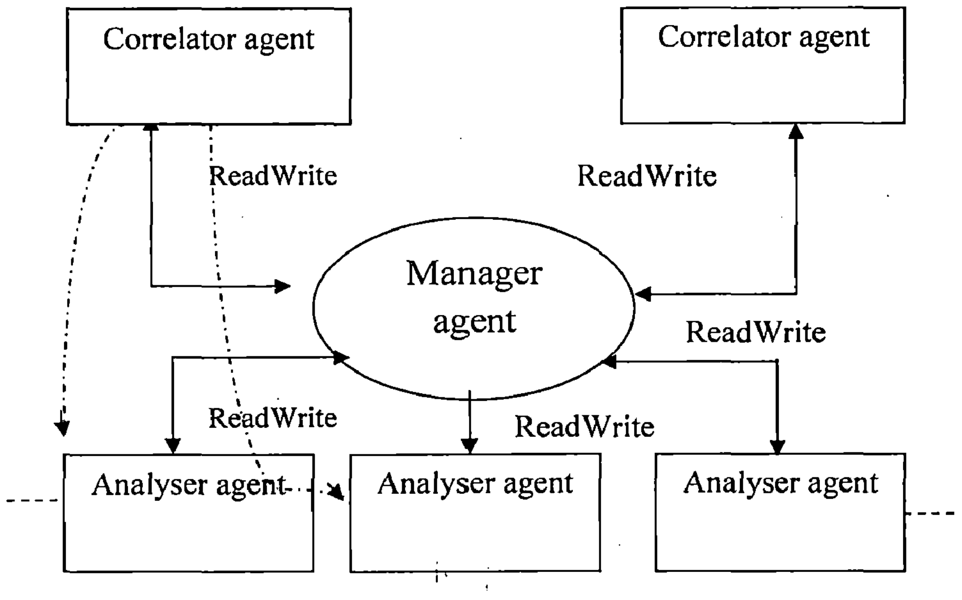


Figure3.3 IMAIDS Architecture

Manager agent: This agent gathers collected information and distributes it to analyser agents.

Analyser agent: Several kinds of analysis such as classical signature detection, anomaly detection

3.4 Micael

Micael[15] consists of the head quarter, the sentinels, the detachments, the auditors, and finally the special agents. Figure 3.4 shows Architecture of this System.

The **Head Quarter (QG)** is a special agent that centralizes the system's control functions. It's also responsible by creation the other agents, maintaining this way a database of agents' executable codes. Periodically, the QG creates *auditor* agents, to verify that the whole of the system remains it's integrity.

Sentinels are special agents that remain residents in each of the target network hosts, collecting relevant information, and informing the QG about eventual anomalies, just for logging. When a Sentinel detects an arbitrary level of anomaly, it requests the creation of a **Detachment** to the QG, so the Detachment can verify with greater detail the detected anomaly. Periodically, the sentinel saves its execution state to the QG, preventing abrupt host system's failures or shutdowns.

Auditor: Check the code that detachments executing is correct or not through the database that is maintained at HQ.

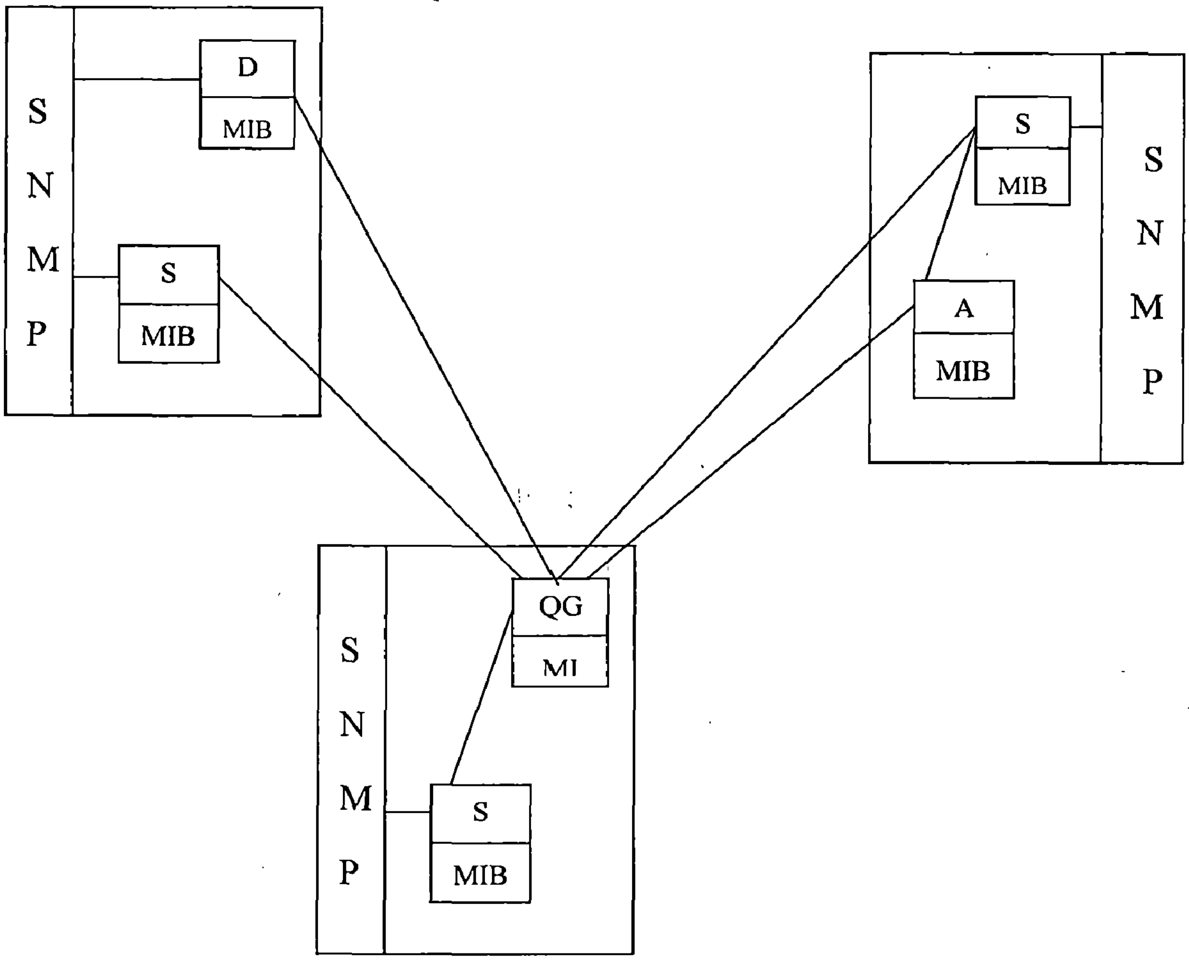


Figure3.4 Micael Architecture

3.5 IA_DIDS (Intelligent Agents for Distributed Intrusion Detection System) [16]

This system architecture is showed in Figure 3.5.

The **Specialized Local Agent** is the engine component of this system. It must combine several kinds of attack analysis such as signature detection, anomaly detection and performed global analysis, for detecting distributed attacks. SLA delegates performed tasks to well defined agents and uses different data sources. As shown in figure SLA delegates predetermined performed tasks to four agents (Filter, Analyser, Correlate, Interpreter and Mobile), and use two knowledge database (Event Rules, Events DB).

Filter Agent is agent responsible for filtering specialized security events from the log files. It examines the packets for well-known attack events and stores all its characteristics into Event DB. Filter agent uses the rules in the event rules database for filtering.

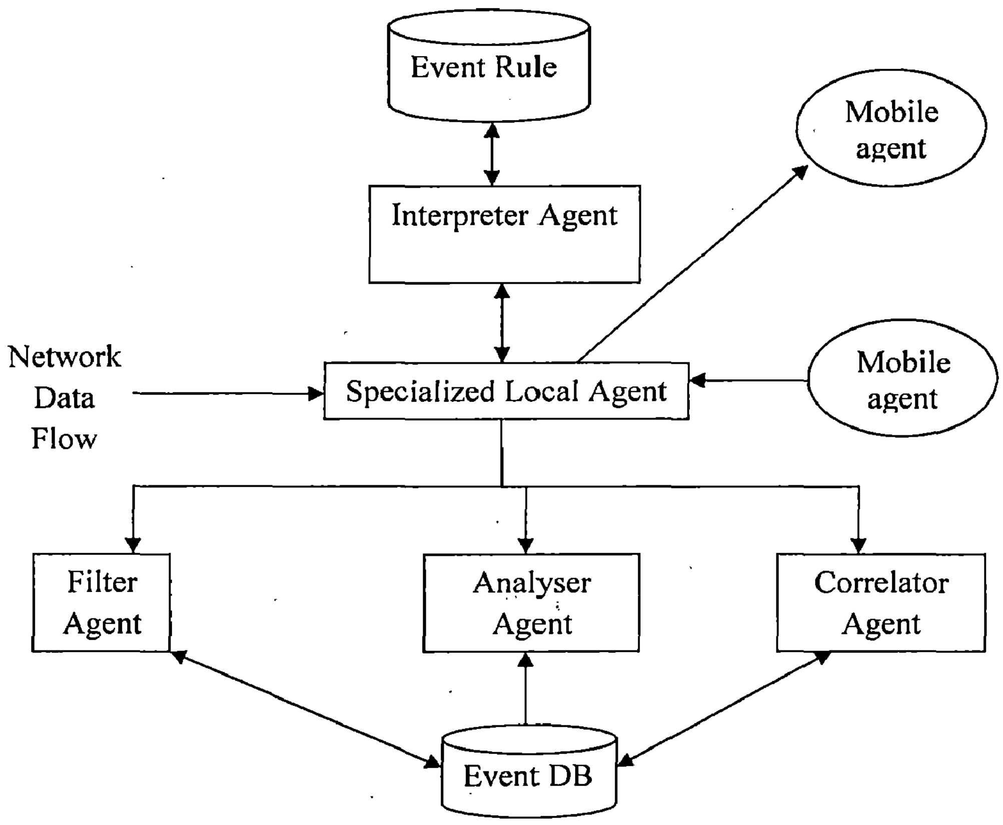


Figure3.5 IA_DIDS Architecture

Analyser Agent analyses the events database. It looks for the local events selected by the *Interpreter Agent*. These patterns are retrieved from *Events DB*. Then, it reports a search results to the *Interpreter Agent* using its Specialized Local Agent.

Interpreter Agent: It collaborates with the *Analyser Agent* for detecting complex local attacks, and uses the *Correlate agent* with the *Mobile Agent* for determining whether some suspicious activities in different node can be combined to be a distributed intrusion.

Correlate agent is responsible for determining whether some suspicious activities in different network nodes can be combined to be a distributed intrusion.

3.6 Research Gaps Identified:

- In AAFID , monitors are single points of failure. If a monitor stops working, all the transceivers that it controls stop producing useful information. This can be solved through a hierarchical structure where the failure of a monitor would be noticed by higher level monitors, and measures would be taken to start a new monitor and examine the situation that caused the original one to fail. Another possibility is to establish redundant monitors that look over the same set of transceivers so that if one of them fails, the other can take over without interrupting its operation. Detection of intrusions at the monitor level is delayed until all the necessary information gets there from the agents and transceivers. This is a problem common to distributed IDSs. Till now there is no AAFID implementations that solves the failure of monitors through redundant monitors.
- The main disadvantage of the Intrusion Detection Agent(IDA) is scalability, because managers can deal with only a limited amount of sensors and agents. Till now there is no IDA implementations that solve the problem of scalability.
- In IMAIDS Detection of intrusions at the analyser agent is delayed until all the necessary information gets from the correlator agents. This is a problem common to distributed IDSs. And also correlator agent has to maintain some rules to find the crucial events that are derived from set of simple events. It is difficult to update rules that are maintained by correlator agent.
- The Intelligent Agents for Distributed Intrusion Detection System has to maintain event database and event rules at each host. This event rule base consists of some events that correlate to intrusion. It has to be updated according to the

environment. This is a very difficult task. Till now there is no successful implementations developed based on this concept.

Due to the above gaps, we have been motivated to propose a novel agent based Distributed Intrusion Detection System.

4. PROPOSED IDS

In this chapter, the proposed IDS system architecture is discussed in detail. The IDS consist of a distributed IDS integrated with mobile agents. The IDS detect network intrusion from outsiders as well as from insiders. It also trace backs the origin of the attack, if the attack is generated from inside the network. In general, it is difficult to find the point of origin of the intrusion in internal network by IDS. The reason is that the internal attacker, the authorized user, uses various tools to find internal network information like IP address of the internal network hosts, which ports are opened in that host. With the knowledge gained by internal attacker, internal attacker generates attack with spoofed IP addresses. To overcome this disadvantage, the architecture presented uses mobile agents along with a NIDS to trace internal attacks. This is shown in Figure 4.1.

4.1 Proposed System Architecture

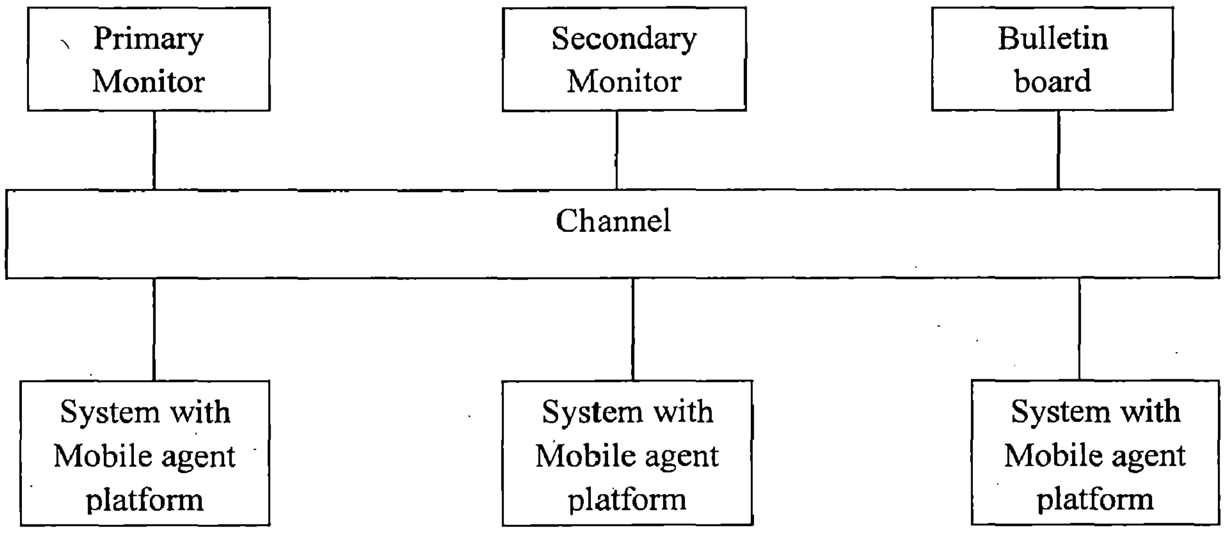

Primary Monitor
Secondary Monitor
Bulletin board
Channel
System with Mobile agent platform
System with Mobile agent platform
System with Mobile agent platform

Figure 4.1 Overall System Architecture.

In the proposed system, a primary monitor station is replicated to remove single point of failure and bottlenecks. In some cases, if at all primary monitor fails to detect an attack, then there is chance to detect that attack by a secondary monitor. Secondly, this approach provides highly distributed IDS that reduce the traffic in the network by using local processing units (static agents) to analyze relevant data and send summaries to monitor station.

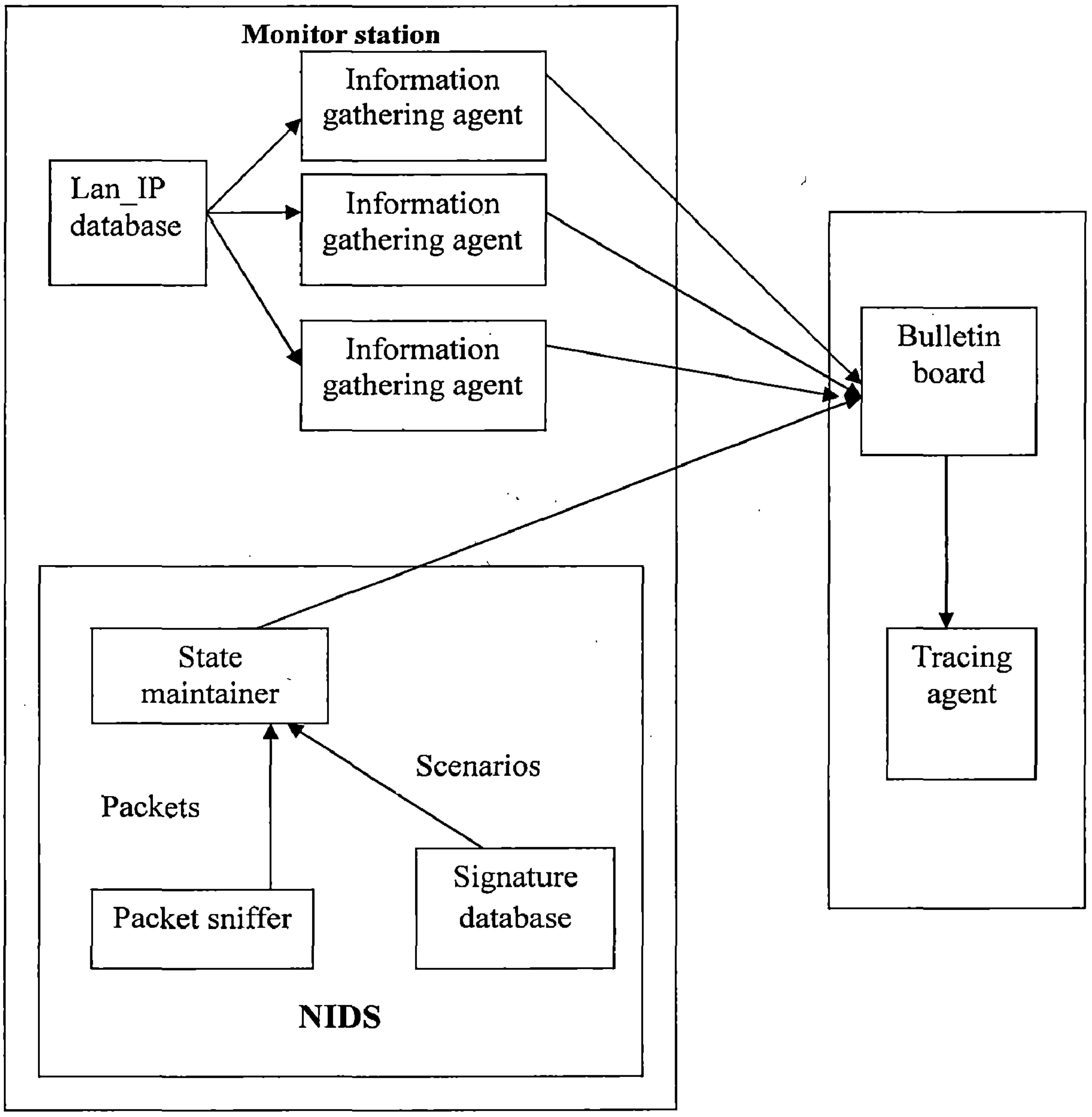


Figure 4.2 Proposed IDS

Figures 4.1 and 4.2 give the block diagrams of the Proposed System. The main components in it are:

➢ Monitor Stations

- Bulletin Board
- Mobile Agent Platform

4.1.1 Monitor Station: Monitor Station is the place where one analyzes the network traffic to find the signs of attack. To increase the detection rate and to eliminate single point failure in the IDS it is possible to place more than one monitor station at different key positions in the network. All the Monitor Stations cooperate with each other to find the attack. A Monitor Station consists of the following components.

- Packet Sniffer
- State Maintainer
- Signature Database

Figure 4.3 shows the Monitor station.

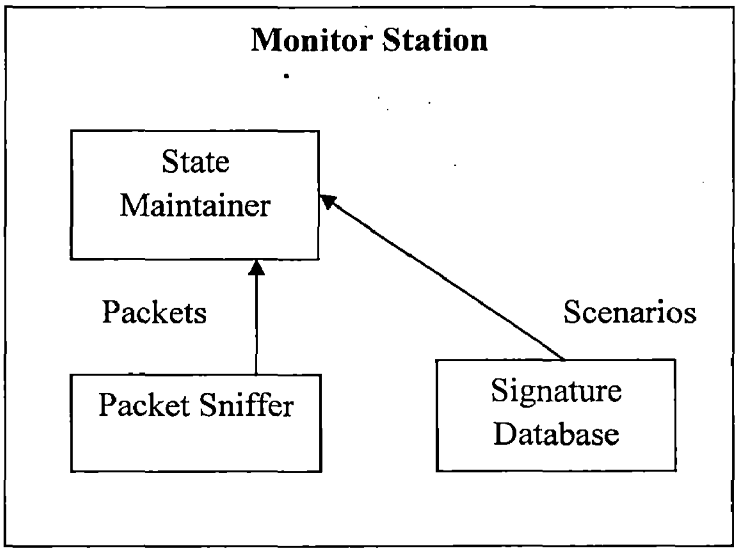


Figure 4.3 Monitor Station

4.1.1.1 Packet Sniffer: A sniffer is a device used to tap into networks to allow an application or hardware device to eavesdrop on network traffic. The traffic can be ARP, IP, TCP, UDP, ICMP packets. The Packet Sniffer that is maintained at the primary monitor is used for reading packets in the network by setting Network Interface Card in promiscuous mode. In this model, Packet Sniffer is continuously running at each Monitor Station.

To reduce the load on the sniffer it is also possible to assign filters to sniffer. For example, if one wants to see only TCP packets for attacks, filter can be set such that all UDP, ICMP, ARP packets are eliminated.

4.1.1.2 State Maintainer: State Maintainer creates the Attack Scenario from Signature Database. After reading the packet, Sniffer sends the packet to State Maintainer. State Maintainer analyzes the packet for any possible attack. If it finds any attack it sends all the required information about that particular attack to Bulletin Board for further processing.

4.1.1.3 ***Signature Database:*** The misuse detection model is used in Proposed System. Therefore, the Signature Database is maintained at Monitor Station. This database consists of signatures of various attacks. This database is designed such that it can be updated whenever any new attack is found. The details about the signatures are given in section 4.3.

4.1.2 Bulletin Board

Bulletin Board is a place where administrator interacts with the IDS. After finding the signs of an attack, Monitor Stations inform to Bulletin Board. Monitor Stations also send the detailed information about the attack. The information consist of the packets those signs matches to the signs of an attack, packet received time. It also receives suspicious activities at different systems in the network that are gathered by Information Gathering Agent. It indexes all this information in such a manner that it can find information quickly when ever Tracing Agent requests for it.

4.1.3 Mobile Agent Platform

Mobile Agents are used to gather suspicious activities at different hosts and also to trace the origin of the attack if the attack is from inside the network. In the proposed IDS, mobile agent platform consists of 3 agents and a database.

- Static Agent
- Information Gathering Agent
- Tracing Agent

- Lan_IP Database

4.1.3.1 ***Static Agent:*** A Static Agent is running in each host in the LAN. It observes the suspicious activities at the host and stores them separately from system log. This separation helps the Information Gathering Agent to gather required information quickly.

The Static Agent also sends periodic ALIVE messages to other Static Agents. This periodic message helps to estimate down times of different systems. This information helps while tracing the attack. It is possible for an attacker to turn off the mobile agent platform for a particular amount of time and launch the attack. This type of behavior is also considered as suspicious activity. The different suspicious activities for the generation of network intrusion include:

- Using local system administration privileges
- Creation of RAW sockets
- Turn off the mobile agent platform

For recording the usage of local system administration privileges inotify-java, a kernel sub system for Linux has been used.

For gathering information about RAW socket creation, the Linux kernel has been modified in such a way that it logs all the RAW socket creations. Static Agent reads this log and stores them separately with information collected from inotify.

The information about shutdown of mobile agents platform is gathered using periodical ALIVE messages.

4.1.3.2 Information Gathering Agent: Monitor Stations and Bulletin Board periodically send Information Gathering Agents to different hosts in the LAN. Before going to the host, Information Gathering Agents gather information about that host from Bulletin Board. The information includes the time of last Information Gathering Agent to this host and possible downtimes of agent platform in the host.

As and when Information Gathering Agent dispatches to host, Information Gathering Agent summaries the file maintained by Static Agent and sends the summary to Bulletin

Board. It also gathers any suspicious activities from system log for the downtimes. Even though Mobile Agent Platform is down, system logs still consist of suspicious events like RAW socket creation.

4.1.3.3 Tracing Agent: At Bulletin Board one Tracing Agent runs. This Tracing Agent gets the summaries sent by Information Gathering Agents and the results sent by the NIDS from the Bulletin Board. After getting the required information from the Bulletin Board, Tracing Agent analyses the information to find the point of origin of the attack. In this work, NIDS are tested with the attacks that are dependent on the RAW socket generation. Therefore to find the point of origin of an attack, Tracing Agent compares the RAW socket generated time with the results that are sent by NIDS. If the match is found then, Tracing Agent concludes there is a presence of attack in the network.

4.2 Working of Proposed System

- At each host in the LAN, a Static Agent runs.
- Static Agent first gets the suspicious activity list that is maintained by Monitor Station.
- In this work, the suspicious activity list consists of the RAW socket generation and use of local system administration privileges and down times of agent platform.
- If the static agent detects activities in the suspicious activities present in the list, then it stores the observed information in one log.
- The monitor station creates Information Gathering Agent for each host in the LAN.
- Dispatch the Information Gathering Agents to each host in the LAN.
- Information Gathering Agent analyses the file that is maintained by Static Agent.
- Summaries of Information Gathering Agents sent to Bulletin Board.
- At Monitor Station, one NIDS based on misuse detection mode is executed. Therefore one database, which consists of signature of various attacks is maintained at Monitor Station.
- NIDS reads the packets in the network by setting network interface in promiscuous mode and compare these packets with the signatures that are present in the database.

- If the match is found, NIDS concludes that there is a presence of attack and sends the necessary information to the Bulletin Board.
- At each Monitor Station, there is one tracing agent. This tracing agent gets the information present in the Bulletin Board.
- After getting necessary information from the Bulletin Board, Tracing Agent analyses the information to find point of origin of an attack.

4.3 Attack Signatures

An Attack Signature is a unique arrangement of information that can be used to identify an attacker's attempt to exploit a known operating system or application vulnerability [17].

The proposed system uses one standard (AISF) for signature of an attack [18]. According to this standard, signature of attack consists of various modules; these are Signature Identification Module, Signature Information Module, Signature Characteristics Module, IP Module, Data Link Module, TCP Module, UDP Module and ICMP Module. IP, ICMP, Data Link, TCP and UDP module consists of respective protocols information for the attack.

Signature Identification Module

In this module there are following fields

Version: field devised for the identification of the AISF model.

ID: identifies the attack. It is a unique number.

Name: this is common name of the intrusion event.

Next Module: identifies next module following this module in the signature of an attack.

In our work we have used EOF in Next Module to specify that module is end of module for that attack.

Signature Information Module

In this module there are following fields

Security level: describes how dangerous this attack.

Category: Intrusion event category, like scan, Dos, or interactive attack.

Description: describes the cause of this attack.

Impact: What is the consequence of this attack.

Target system: System that are more commonly affected by this attack.

Next module: the same as described above.

Signature Characteristics Module

In this module there are following fields

Ease of Attack: describes how easily this attack can be realised.

Recommended Actions: recommended preventive an /or corrective actions to take.

Threshold: count for number of events of this type leads to an attack.

Next Module: as described above.

The following modules are easily understandable, representing more technical side of an attack signature. They represent required information for intrusion detection, like data link, network and transport layers.

Data Link Module

This module consists of Source Address, Destination address and Next Module.

IP Module

This module consists of Packet length, Type of service, Fragment ID, Flags, Fragment Offset, TTL, Source Address, Destination Address, Options, Protocol, Expression and Next Module.

TCP Module

This module consists of Packet length, Source Port, Destination Port, Sequence Number, Acknowledge Number, Data offset, Flags, Window, Urgent pointer, Options and Next Module.

UDP Module

This module consists of Packet length, Source Port, Destination Port and Next Module.

ICMP Module

This module consists of Packet length, Type, Code, ID, Sequence and Next Module.

We have followed some rules for representing the signature of an attack

i. na to represent, that attribute not applicable,

ii. lt to represent less than

iii. gt to represent greater than

iv. eq to represent equals to

v. neq to represent not equals to

vi. S to represent stored

vii. s to represent store

viii. con represent contains

ix. exp represent expression

x. EOF represent this module is end module for this attack

5. IMPLEMENTATION

This chapter gives the implementation details of the proposed system and discusses results.

5.1 Monitor Station

As discussed in chapter 4, Monitor Station consists of the fallowing components.

1. Packet Sniffer
2. State Maintainer
3. Signature Database

To implement Packet Sniffer we have used jpcap 0.7. This is an open source library. Figure 5.1 shows the interface between Sniffer and State Maintainer.

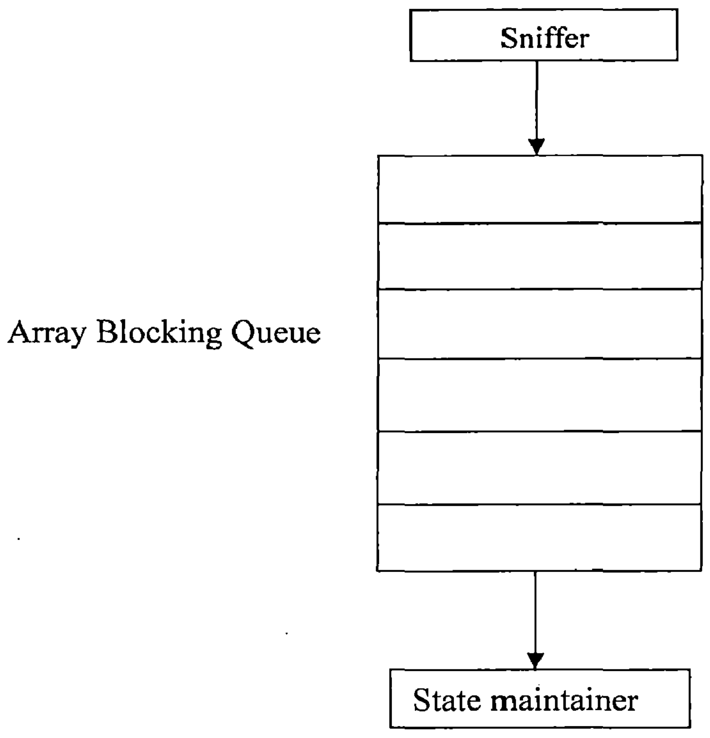


Figure 5.1 Interface between sniffer and state maintainer

After reading the packet, the Packet Sniffer puts the packet in the queue between Sniffer and State Maintainer. State maintainer reads each packet from the queue and gives it to the appropriate maintainers. For example if the packet is of type IP then it gives the packet to IPMaintainer. If the packet is of type TCP it gives it to the TCPMaintainer. Figure 5.2 shows State maintainer.

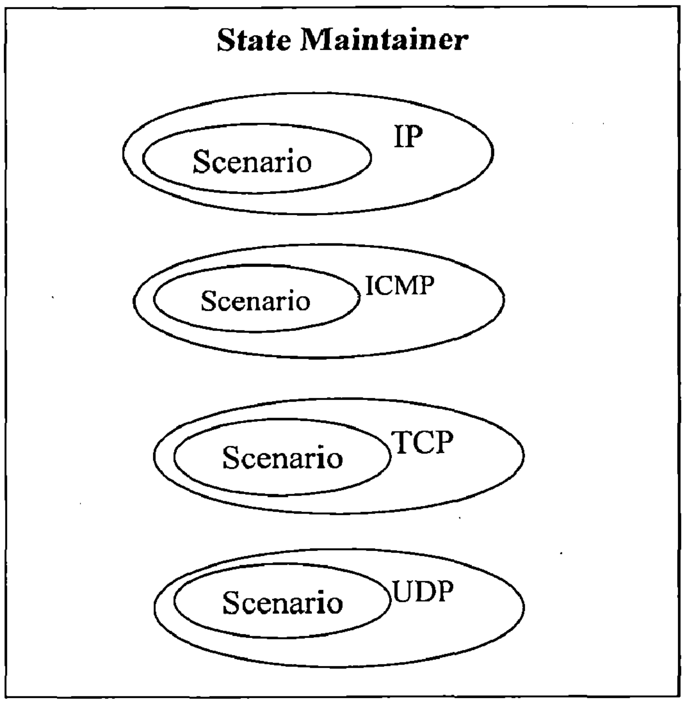


Figure 5.2 State Maintainer

State Maintainer consists of IP, TCP, ICMP, UDP maintainers. These maintainers consist of different scenarios. Each scenario represents one attack signature.

5.2 Creation of Scenarios

For each attack signature present in the signature database we have created the scenario. Scenario contains rule table, store table along with attack signature information. Rule table contains rules for checking that corresponding attack. Store table contains rules for when to store, if these rules matched then what are the values to store for further checking attack. After successful creation of scenario, monitor station add that scenario to corresponding maintainer depending on end module of an attack signature. For example if attack signature's end module IP then created scenario is added to IP maintainer.

We have divided all the signatures in the signature data base into two types. First type of signatures is those which only depend on single packet. For example, consider the attack "ping of death". In ping of death attack, we can find the attack if we receive a single packet with length more than 65535 bytes. The signature data base for this type of attack is as follows.

Signature Identification Module

Version	:	0.7
ID	:	1
Name	:	ping of death
Next Module	:	SCM

Signature Information Module:

Security level	:	medium
Category	:	dos
Description	:	ICMP packet size > 65535
Impact	:	system will not respond
Target system	:	Win 95, Win NT
Next module	:	SCM

Signature Characteristics Module:

Ease of Attack	:	true
Recommended Actions:		simply reboot
Threshold	:	1
Next Module	:	IP

IP Module

Packet length	:	na
Type of service	:	na
Fragment ID	:	na
Flags	:	na

Fragment Offset	:	gt 8192
TTL	:	na
Source Address	:	na
Destination Address	:	na
Options	:	na
Protocol	:	na
Expression	:	na
Next Module	:	EOF

Rule table consists of only single entry for this attack signature

0 : IP Fragmentoffset gt 35565

There are no entries for store table because there are no rules that start with s. There is only one rule to match this attack. Matching a packet to see if this type of attacks is came is simple. When we receive a packet compare the rule table to see all the rules are matched. This type of attack does not depend on the store table.

The second types of attacks are those that depend on rule table and store table. For example consider the "syndrop" attack. This depends on two conjunctive packets.

Signature Identification Module

Version	:	0.7
ID	:	1
Name	:	syndrop attack
Next Module	:	SCM

Signature Information Module:

Security level	:	high
Category	:	dos
Description	:	overlap of IP fragments
Impact	:	system will not respond
Target system	:	Win 95, Win NT
Next module	:	SCM

Signature Characteristics Module:

Ease of Attack	:	false
Recommended Actions:		simply reboot
Threshold	:	1
Next Module	:	IP

IP Module

Packet length	:	s con $ip.flags MF
Type of service	:	na
Fragment ID	:	s con $ip.flags MF
Flags	:	na
Fragment Offset	:	lt S $ip.PacketLength
TTL	:	na
Source Address	:	na
Destination Address	:	na
Options	:	na
Protocol	:	na
Expression	:	exp eq $ip.FragmentID s $ip.FragmentID
Next Module	:	EOF

Scenario's rule table generated by primary monitor for this attack consists of

a. IP Fragment Offset lt S $ip.packetLegth
b. IP Expressio exp eq $ip.FragmentID s $ip.FragmentID

Scenario's store table generated by primary monitor for this attack consists of

a. Rules
   i. con $ip.flags MF
b. Values
   i. Fragment ID
   ii. PacketLength

To match this type of attack the fallowing procedure is followed.

1. When a packet comes, first check to see if all the rules in store table are matched.
2. If yes then store this packet in a Array List.
3. If no match the packet with the rules in the rule table.
4. If the rule starts with "s" then the field value needs to match with a packet stored in ArrayList.
5. To get such type of packets create temporary ArrayList. Store all the packets from original store ArrayList.
6. Match all the conjunctive rules with this temporary list.
7. If all the rules in the rule table matched then a attack has happend.

5.3 Mobile Agents

Implementation of mobile agents in languages such as JAVA provides them with system and platform independence and considerable security features, which are a necessity in Intrusion Detection Systems. Therefore we have implemented mobile agents with ibm aglets2.0.12.

5.3.1 Static Agent:

At each host in the intranet (LAN), we run one Static Agent, which looks for information related to malicious activity in that host. For getting malicious activity we have maintained one suspicious activity list. This list consists of events that are observed by static agent. In our work, this list consists of RAW socket generation and password file modification. After getting suspicious list this Static Agent observes the activities that are present in that list.

For observing RAW socket generation, we have modified the kernel such that when RAW socket is generated it logs that socket information, for modifying kernel we have followed [19]. By reading log file Static Agent get RAW socket information.

According to [19], we have inserted the following code in socket.c's sys_socket(int family, int type, int protocol) method.

```
if(type==3)
        printk(KERN_DEBUG "\n my kernel ... A socket is created with family %d, type
%d, protocol %d and process %d \n",family,type,protocol,current->pid);
```

This code simply logs socket information, when RAW socket is generated by the host. In our work, we also have redirected log messages to inotify.txt file with klogd, which is a system daemon which intercepts and logs Linux kernel messages. For redirecting log messages to inotify.txt, first we have to avoid auto-backgrounding of klogd. This is needed especially if the klogd is started and controlled by init(8), for that we have used -n option for klogd daemon. After that we have used –f option to send log messages to the inotify.txt file.

For observing password file modification we have used inotify-java API utility [20]. The inotify-java API provides an event-based mechanism for monitoring Linux file system events using the inotify interface provided by glibc (versions 2.4 and up) and the Linux kernel, starting from 2.6.13.

Inotify is an inode-based file notification system that does not require a file ever be opened in order to watch it. inotify does not pin filesystem mounts—in fact, it has a clever event that notifies the user whenever a file's backing filesystem is unmounted. inotify is able to watch any filesystem object whatsoever, and when watching directories, it is able to tell the user the name of the file inside of the directory that changed[20].

For monitoring password file modification, we have simply created instance of inotify-java API's Inotify class. After that we have added InotifyEventListener to the Inotify class. In InotifyEventListener we have implemented filesystemEventOccurred(InotifyEvent event) method for IN_MODIFY event. This is a sample code for monitoring password file system modification by static agent. Static agent stores observed suspicious activity information along with time when that suspicious activity occurred in that host into the inotift.txt file in our work. That observed information may be RAW socket generation or it may be password file modification.

```
myfile = new FileWriter("inotify.txt", true);
out = new BufferedWriter(myfile);
Inotify mynotify = new Inotify();
InotifyEventListener e= new InotifyEventListener()
{
   public void filesystemEventOccurred(InotifyEvent event)
    {
      if(event.IN_MODIFY())
      {
        out.newLine();
        out.write(" my kernel modify occured ......");
      }
    }
};
mynotify.addInotifyEventListener(e);
mynotify.addWatch("/etc/pwd", Constants.IN_ALL_EVENTS);
```

In above code mynotify.addWatch("/etc/pwd", Constants.IN_ALL_EVENTS) adds a watch to filename pwd file. General syntax of addWatch method is given below addWatch("filename", Constants.MASK), where MASK is one of ACCESS, MODIFY, ATTRIB, CLOSE_WRITE, CLOSE_NOWRITE, OPEN, MOVED_FROM, MOVED_TO, CREATE, DELETE, DELETE_SELF, UNMOUNT, Q_OVERFLOW, IGNORED, ISDIR, ONESHOT, CLOSE, MOVE or ALL_EVENTS.

5.3.2 Information Gathering Agent:

At Monitor Station we have maintained a Lan_IP Database, which consists of IP addresses of hosts in the LAN. For each IP addresses present in the Lan_IP Database Monitor Station creates Information Gathering Agent and dispatched to the corresponding IP address.

We have implemented that logic in IGProxy aglet. First in IGProxy, we have connected to Lan_IP Database and stored all IP addresses in one vector (addresses). For each entry in addresses vector, IGProxy creates the Information Gathering Agent. Following code shows the creation of Information Gathering Agent and dispatch to the addresses present in the Lan_IP by IGProxy.

```
AgletProxy inf_proxy;
inf_proxy=context.createAglet(this.getCodeBase(),"InformationGathering",this.getAgletI
D());
URL url =new URL("atp://"+ipaddresses.get(i)+":4434");
inf_proxy.dispatch(url);
```

As and when Information Gathering Agent dispatched to IP address, Information Gathering Agent summaries the file (inotify.txt) maintained by Static Agent and send the summary to Bulletin Board along with IP address of that system in one string form.

5.3.3 Tracing Agent

We have a Tracing Agent at the Bulletin Board. This agent continuously gets the information present at Bulletin Board to find the point of origin of attack in LAN. For that in our work, we simply compared the time when RAW socket generated in each host with the time when attack is generated. The reason for the same is that, we are tested our work with attacks that depends on RAW socket generation. If the match is found then, tracing agent concludes there is a presence of attack in the network.

5.4 Bulletin board:

In our work, we are run the Bulletin Board in Linux system. i.e. we are run server socket at 4324 port in local host. The following code segment shows the creation of server socket.

```
ServerSocket sc=new ServerSocket(Integer.parseInt("4324"))
```

If the Bulletin Board successfully binds to its port, then the ServerSocket object is successfully created and the Bulletin Board continues to accepting connections from clients. Following code segment shows the accepting connections from clients.

```
while(true)
        {
                Socket client=sc.accept();
        }
```

The accept method waits until a client starts up and requests a connection on the host and port of this bulletin board. When a connection is requested and successfully established, the accept method returns a new Socket object which is bound to the same local port and has it's remote address and remote port set to that of the client. The Bulletin Board can communicate with the client over this new Socket and continue to listen for client connection requests on the original ServerSocket. After the server successfully establishes a connection with a client, Bulletin Board communicates with the client using client's input streams and output streams.

As mentioned earlier, we are sending summaries of Static Agent in string form and NIDS informaton in scenario form. Bulletin Board read objects from client's input stream. If the object is instance of string then, Bulletin Board stores that result in one HashMap with key set to IP address of system from where that summaries came otherwise we are simply displaying Scenario's information in Bulletin Board.

The Bulletin Board also acts as an interface to the administrator. Administrator can also view the activities that are happening in the LAN. We have created a simple UI for bulletin board in which we can view the events host wise or we can view the events at a particular time period. Figure 5.3 shows a screen shot of our Bulletin board.

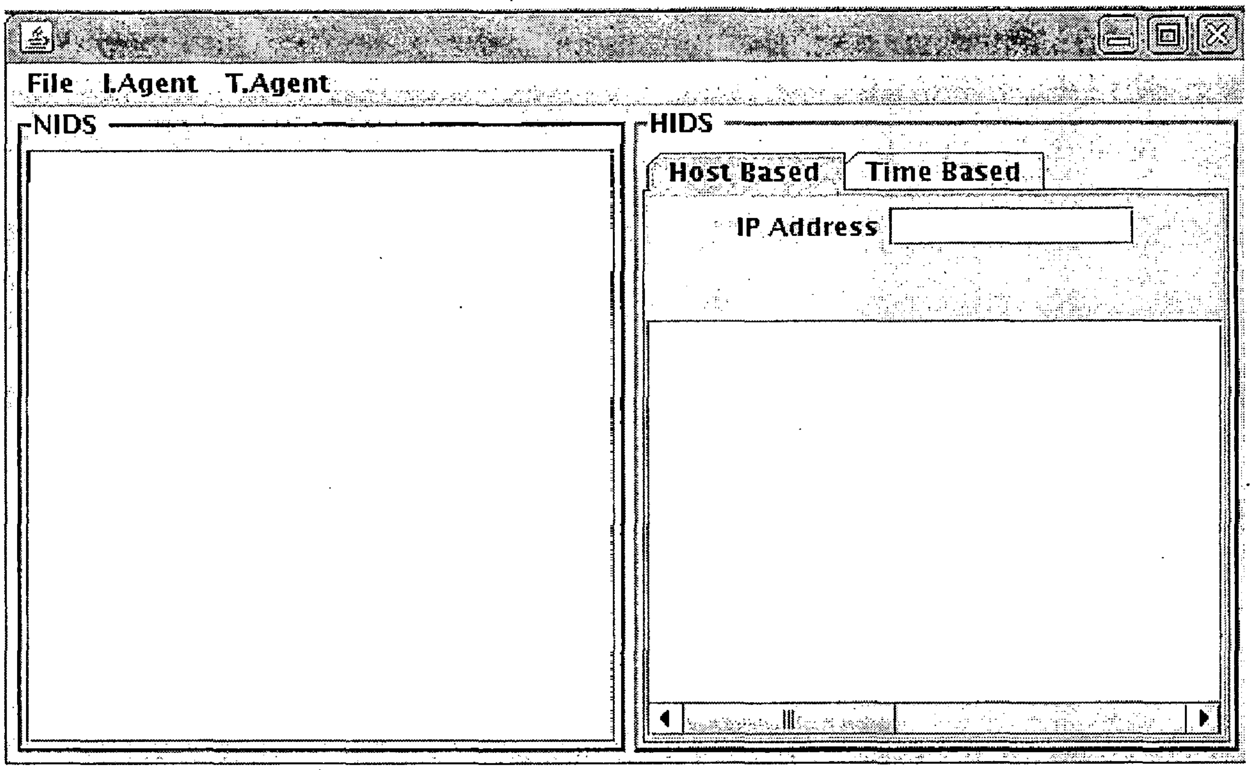


Figure 5.3 Bulletin Board

5.5 Adding and Updating Signature Database

To simplify the adding of new attack signatures and updating and deletion of existing ones, we have created a simple GUI based application which automatically updates all the replicated databases. Figure 5.4 shows the UI of this application.

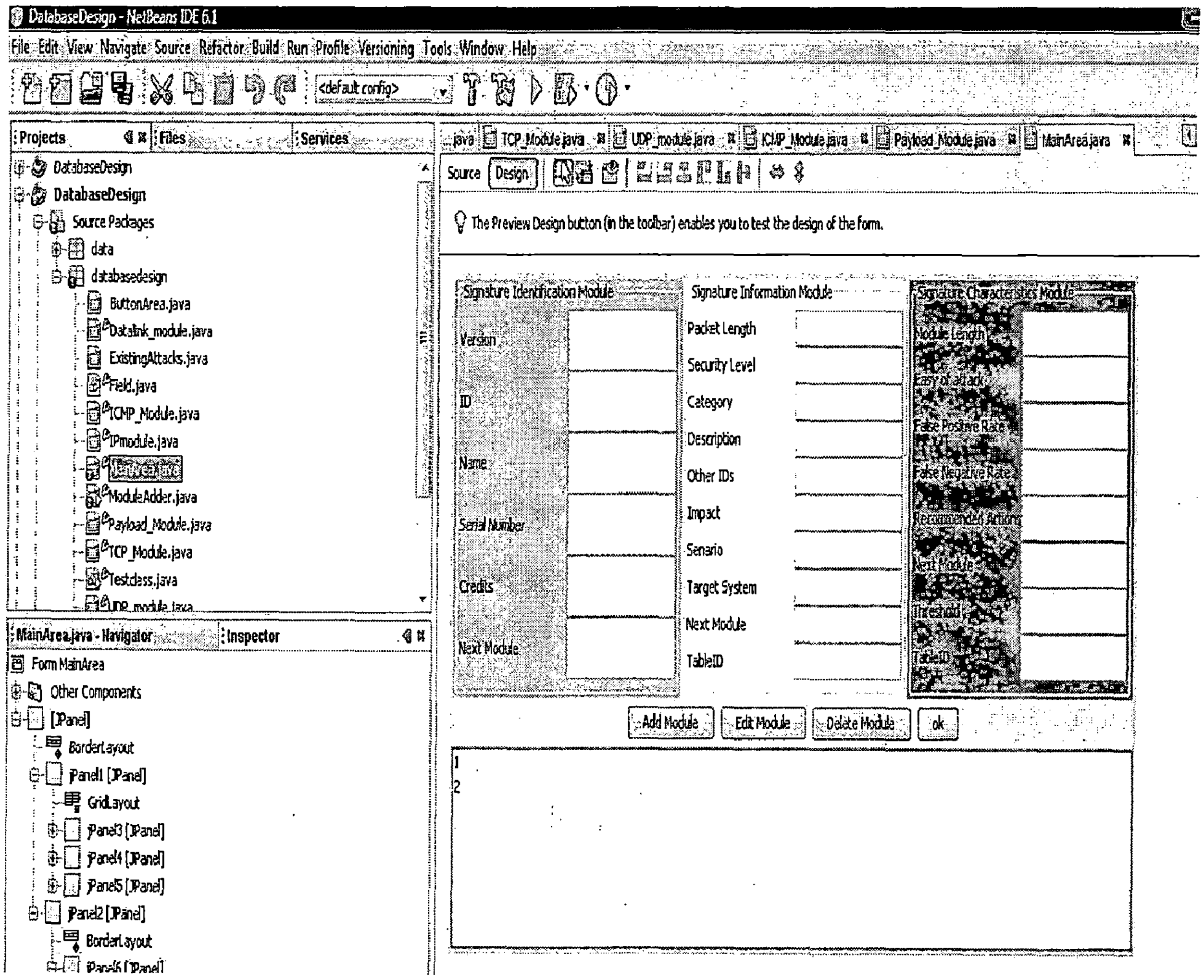


Figure5.4 Signature Database

5.6 Results

The proposed system is tested with 4 attacks these are Teardrop attack, Ping of death attack, Winnuke attack and Syndrop attack. Figure 5.5 shows the Test bed used.

Teardrop attack:

A teardrop attack is a denial of service attack. Some implementations of the TCP/IP IP fragmentation re-assembly code do not properly handle overlapping IP fragments. Impact of this attack is system will not respond.

Ping of death attack:

Ping of Death attack involved sending IP packets of a size greater than 65,535 bytes to the target computer. IP packets of this size are illegal, but applications can be built that are capable of creating them. Carefully programmed operating systems could detect and safely handle illegal IP packets, but some failed to do this. ICMP ping utilities often included large-packet capability and became the namesake of the problem. Impact of this attack is system will not respond.

Winnuke attack:

The WinNuke attack sends OOB (Out-of-Band) data to an IP address of a Windows machine connected to a network and/or Internet. Usually, the WinNuke program connects via port 139, but other ports are vulnerable if they are open. When a Windows NT and Windows 95 machine receives the out-of-band data, it is unable to handle it and exhibits odd behaviour, ranging from a lost Internet connection to a system crash (resulting in the infamous Blue Screen of Death).

Syndrop attack:

This attack is same as the Teardrop, but it generating attack with TCP at as the transport protocol instead of UDP. Impact of this attack is system will not respond.

The proposed system is tested with the network of 6 systems, in that

4 systems : Linux Distributed System,
1 system : Windows 95 (192.168.111.144) and
1 System : Windows NT (192.168.111.146).

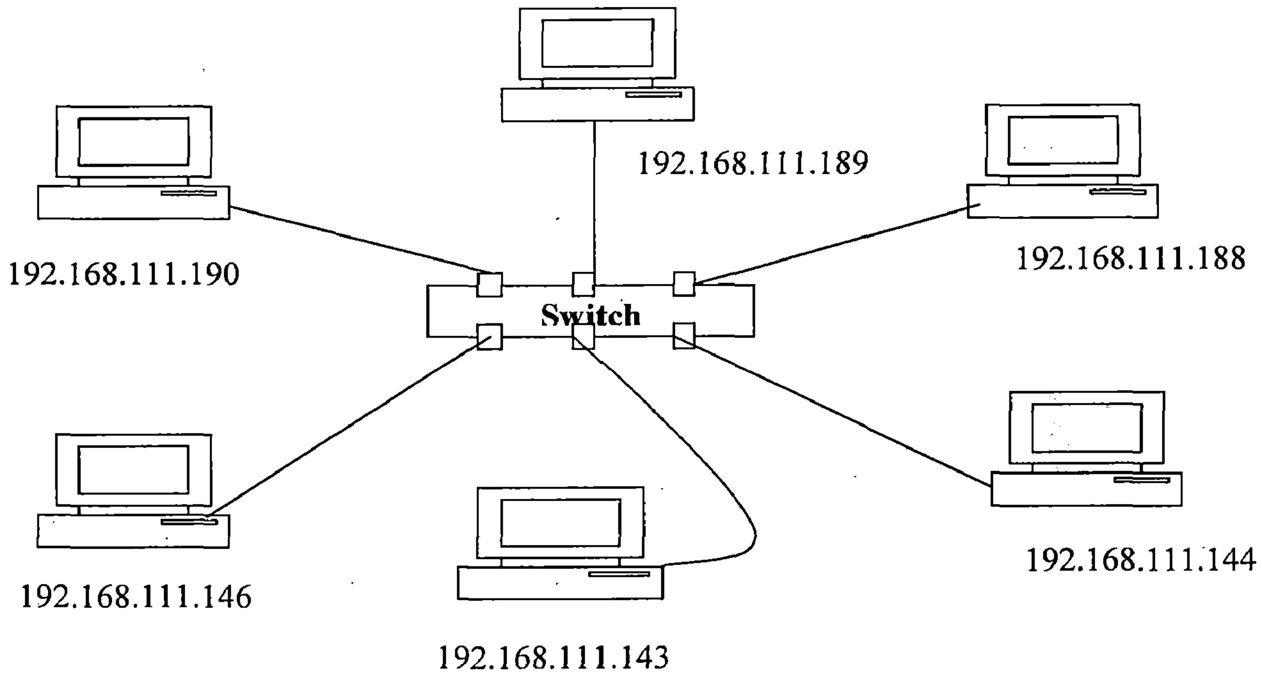
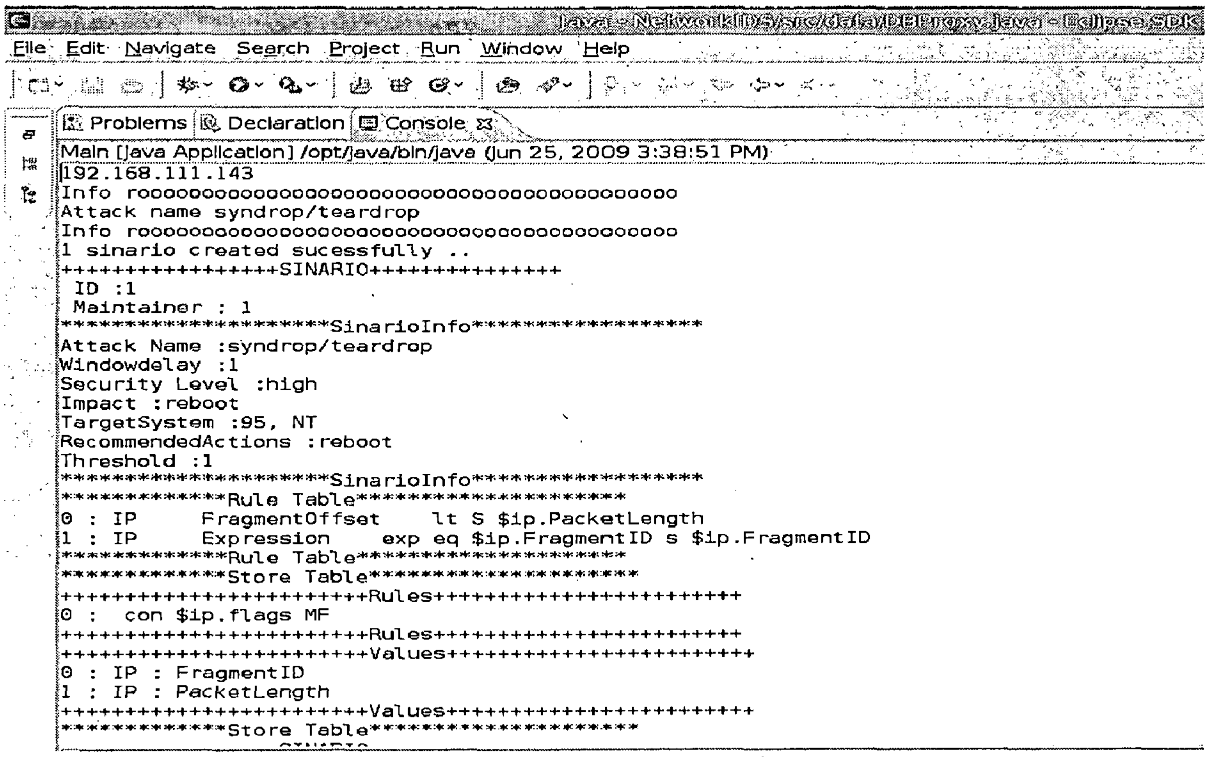


Figure5.5 Test Bed

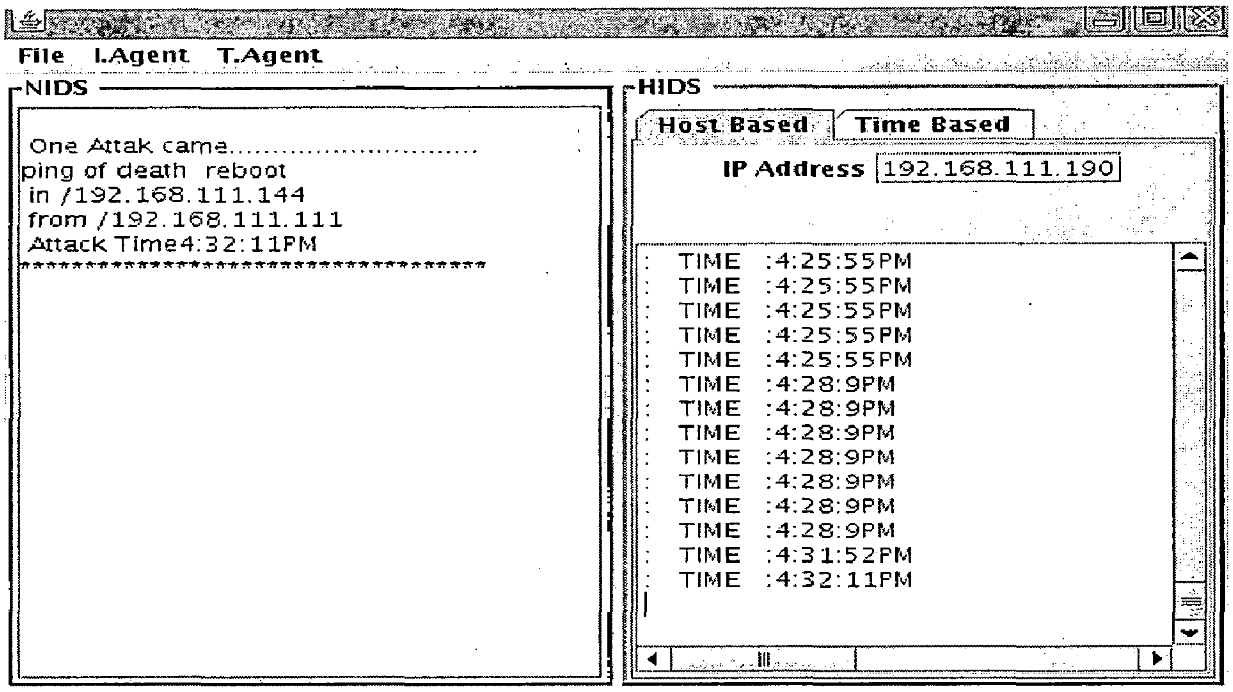
1. To test the proposed system, we run Network IDS and bullet-in board at monitor station. Figure 5.6 shows the snap shot of NIDS.


Java - NetworkIDS/src/data/DBProxy.java - Eclipse SDK
File Edit Navigate Search Project Run Window Help
Problems
Declaration
Console

```
Main [Java Application] /opt/java/bin/java (Jun 25, 2009 3:38:51 PM)
192.168.111.143
Info roooooooooooooooooooooooooooooooooooooooooo
Attack name syndrop/teardrop
Info roooooooooooooooooooooooooooooooooooooooooo
1 sinario created sucessfully ..
+++++++++++++++++SINARIO+++++++++++++++
 ID :1
 Maintainer : 1
**********************SinarioInfo*******************
Attack Name :syndrop/teardrop
Windowdelay :1
Security Level :high
Impact :reboot
TargetSystem :95, NT
RecommendedActions :reboot
Threshold :1
**********************SinarioInfo*******************
*************Rule Table**********************
0 : IP     FragmentOffset    lt S $ip.PacketLength
1 : IP     Expression    exp eq $ip.FragmentID s $ip.FragmentID
*************Rule Table**********************
*************Store Table**********************
++++++++++++++++++++++++Rules++++++++++++++++++++++++
0 :  con $ip.flags MF
++++++++++++++++++++++++Rules++++++++++++++++++++++++
++++++++++++++++++++++++Values++++++++++++++++++++++++
0 : IP : FragmentID
1 : IP : PacketLength
++++++++++++++++++++++++Values++++++++++++++++++++++++
*************Store Table**********************
```

Figure5.6 NIDS at Monitor Station

2. To test the proposed system, we have generated ping of death attack with spoofed IP address 192.168.111.111 through 192.168.111.190. Figure 5.7 shows the Bulletin Board and Figure 5.8 shows Tracing Agent result at that instance respectively.

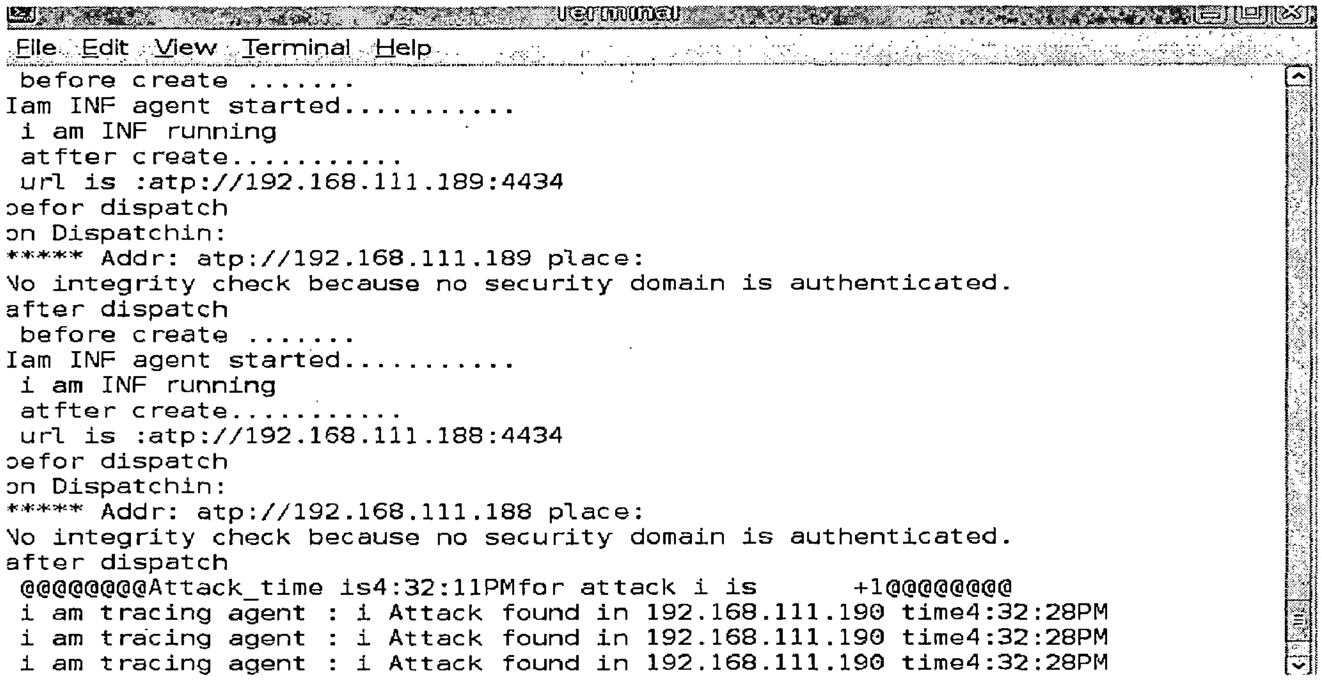


```
One Attak came.............................
ping of death  reboot
in /192.168.111.144
from /192.168.111.111
Attack Time4:32:11PM
************************************
```

IP Address 192.168.111.190

```
:  TIME  :4:25:55PM
:  TIME  :4:25:55PM
:  TIME  :4:25:55PM
:  TIME  :4:25:55PM
:  TIME  :4:25:55PM
:  TIME  :4:28:9PM
:  TIME  :4:28:9PM
:  TIME  :4:28:9PM
:  TIME  :4:28:9PM
:  TIME  :4:28:9PM
:  TIME  :4:28:9PM
:  TIME  :4:28:9PM
:  TIME  :4:31:52PM
:  TIME  :4:32:11PM
```

Figure5.7 Bulletin board

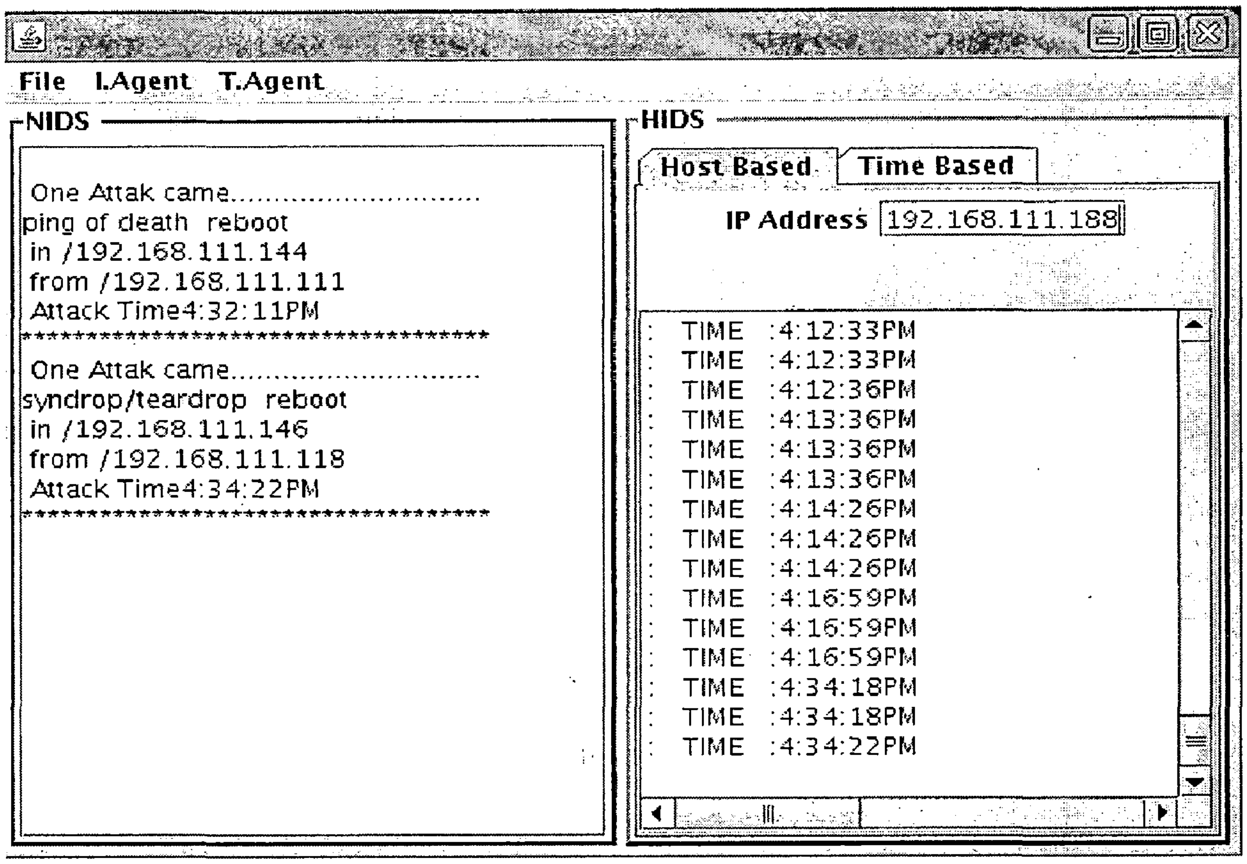


```
 before create .......
Iam INF agent started..........
 i am INF running
 atfter create..........
 url is :atp://192.168.111.189:4434
oefor dispatch
on Dispatchin:
***** Addr: atp://192.168.111.189 place:
No integrity check because no security domain is authenticated.
after dispatch
 before create .......
Iam INF agent started..........
 i am INF running
 atfter create..........
 url is :atp://192.168.111.188:4434
oefor dispatch
on Dispatchin:
***** Addr: atp://192.168.111.188 place:
No integrity check because no security domain is authenticated.
after dispatch
 @@@@@@@@Attack_time is4:32:11PMfor attack i is      +1@@@@@@@@
 i am tracing agent : i Attack found in 192.168.111.190 time4:32:28PM
 i am tracing agent : i Attack found in 192.168.111.190 time4:32:28PM
 i am tracing agent : i Attack found in 192.168.111.190 time4:32:28PM
```

Figure5.8 Tracing Agent

Form Figure 5.7 and 5.8, we conclude that the system successfully traces the point of origin of an attack. This can be used when the administrator wants to take the recommended actions for the system when that particular system generates huge attacks.

Next, to test the system with syndrop attack, we generated syndrop attack at 192.168.111.188 with the spoofed IP address 192.168.111.118. Figures 5.9 and 5.10 show the snap shot of Bulletin Board and Tracing Agent outputs respectively at that instance.

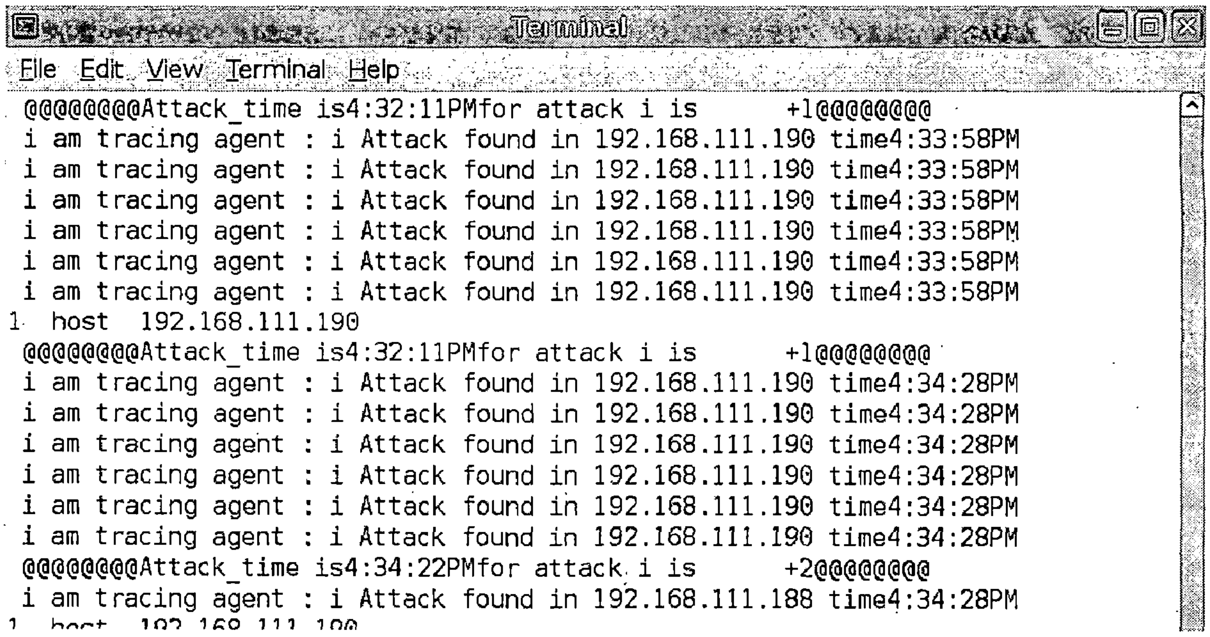


```
One Attak came............................
ping of death  reboot
 in /192.168.111.144
 from /192.168.111.111
 Attack Time4:32:11PM
*************************************
 One Attak came............................
syndrop/teardrop  reboot
 in /192.168.111.146
 from /192.168.111.118
 Attack Time4:34:22PM
*************************************
```

IP Address 192.168.111.188

```
:  TIME   :4:12:33PM
:  TIME   :4:12:33PM
:  TIME   :4:12:36PM
:  TIME   :4:13:36PM
:  TIME   :4:13:36PM
:  TIME   :4:13:36PM
:  TIME   :4:14:26PM
:  TIME   :4:14:26PM
:  TIME   :4:14:26PM
:  TIME   :4:16:59PM
:  TIME   :4:16:59PM
:  TIME   :4:16:59PM
:  TIME   :4:34:18PM
:  TIME   :4:34:18PM
:  TIME   :4:34:22PM
```

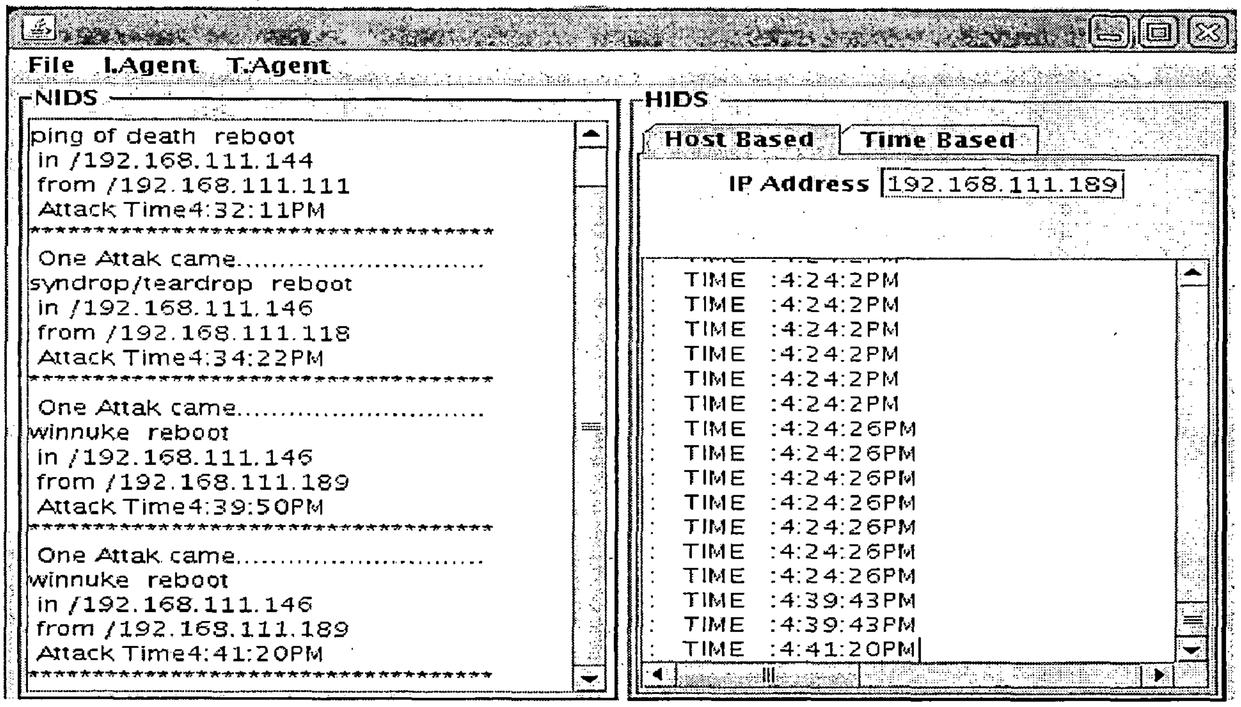
Figure5.9 Bulletin board



```
 @@@@@@@@@Attack_time is4:32:11PMfor attack i is      +1@@@@@@@@@
 i am tracing agent : i Attack found in 192.168.111.190 time4:33:58PM
 i am tracing agent : i Attack found in 192.168.111.190 time4:33:58PM
 i am tracing agent : i Attack found in 192.168.111.190 time4:33:58PM
 i am tracing agent : i Attack found in 192.168.111.190 time4:33:58PM
 i am tracing agent : i Attack found in 192.168.111.190 time4:33:58PM
 i am tracing agent : i Attack found in 192.168.111.190 time4:33:58PM
1  host  192.168.111.190
 @@@@@@@@@Attack_time is4:32:11PMfor attack i is      +1@@@@@@@@@
 i am tracing agent : i Attack found in 192.168.111.190 time4:34:28PM
 i am tracing agent : i Attack found in 192.168.111.190 time4:34:28PM
 i am tracing agent : i Attack found in 192.168.111.190 time4:34:28PM
 i am tracing agent : i Attack found in 192.168.111.190 time4:34:28PM
 i am tracing agent : i Attack found in 192.168.111.190 time4:34:28PM
 i am tracing agent : i Attack found in 192.168.111.190 time4:34:28PM
 @@@@@@@@@Attack_time is4:34:22PMfor attack i is      +2@@@@@@@@@
 i am tracing agent : i Attack found in 192.168.111.188 time4:34:28PM
1  host  192.168.111.190
```

Figure5.10 Tracing agent output

Next, to tests the system with winnuke attack, we generated winnuke attack at 192.168.111.189. Figures 5.11 and 5.12 show the snap shot of Bulletin board and Target System outputs respectively at that instance.

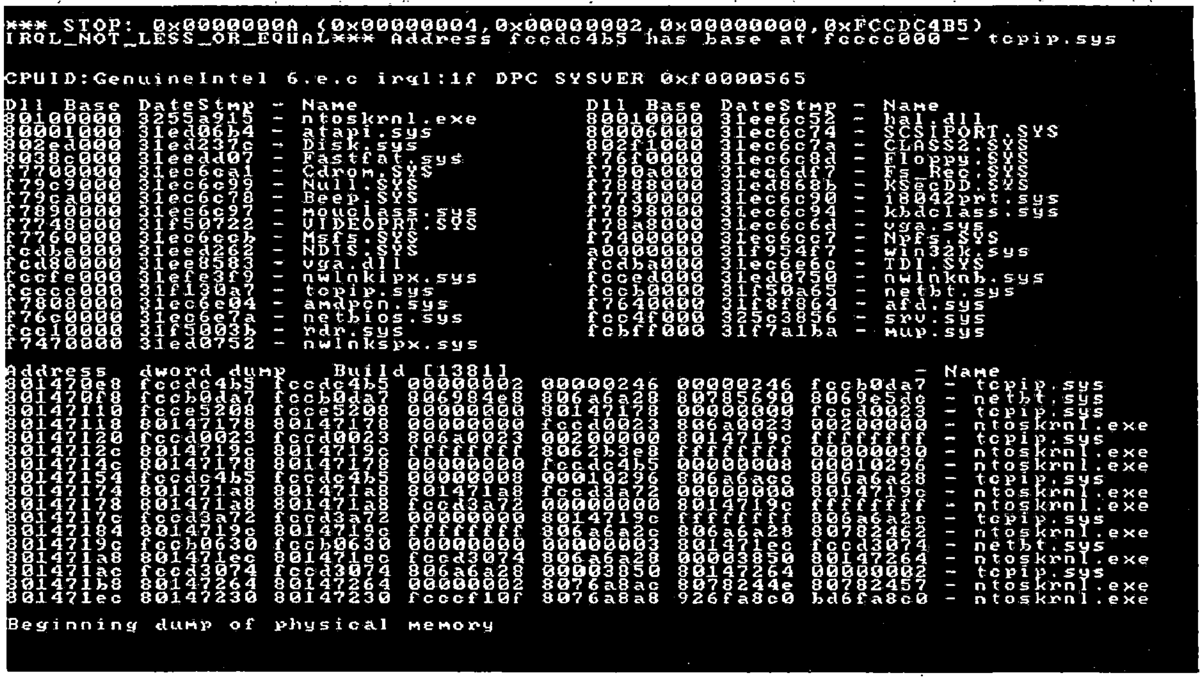


```
ping of death  reboot
 in /192.168.111.144
 from /192.168.111.111
 Attack Time4:32:11PM
************************************
 One Attak came............................
syndrop/teardrop  reboot
 in /192.168.111.146
 from /192.168.111.118
 Attack Time4:34:22PM
************************************
 One Attak came............................
winnuke  reboot
 in /192.168.111.146
 from /192.168.111.189
 Attack Time4:39:50PM
************************************
 One Attak came............................
winnuke  reboot
 in /192.168.111.146
 from /192.168.111.189
 Attack Time4:41:20PM
************************************
```

```
IP Address 192.168.111.189

:   TIME  :4:24:2PM
:   TIME  :4:24:2PM
:   TIME  :4:24:2PM
:   TIME  :4:24:2PM
:   TIME  :4:24:2PM
:   TIME  :4:24:2PM
:   TIME  :4:24:26PM
:   TIME  :4:24:26PM
:   TIME  :4:24:26PM
:   TIME  :4:24:26PM
:   TIME  :4:24:26PM
:   TIME  :4:24:26PM
:   TIME  :4:24:26PM
:   TIME  :4:39:43PM
:   TIME  :4:39:43PM
:   TIME  :4:41:20PM
```

Figure5.11 Bulletin board



```
*** STOP: 0x0000000A (0x00000004,0x00000002,0x00000000,0xFCCDC4B5)
IRQL_NOT_LESS_OR_EQUAL*** Address fccdc4b5 has base at fcccc000 - tcpip.sys

CPUID:GenuineIntel 6.e.c irql:1f DPC SYSVER 0xf0000565

Dll Base DateStmp - Name                     Dll Base DateStmp - Name
80100000 3255a915 - ntoskrnl.exe             80010000 31ee6c52 - hal.dll
80001000 31ed06b4 - atapi.sys                80006000 31ec6c74 - SCSIPORT.SYS
802ed000 31ed237c - Disk.sys                 802f1000 31ec6c7a - CLASS2.SYS
8038c000 31eedd07 - Fastfat.sys              f76f0000 31ec6c8d - Floppy.SYS
f7700000 31ec6ca1 - Cdrom.SYS                f790a000 31ec6df7 - Fs_Rec.SYS
f79c9000 31ec6c99 - Null.SYS                 f7888000 31ed868b - KSecDD.SYS
f79ca000 31ec6c78 - Beep.SYS                 f7730000 31ec6c90 - i8042prt.sys
f7890000 31ec6c97 - mouclass.sys             f7898000 31ec6c94 - kbdclass.sys
f7748000 31f50722 - VIDEOPRT.SYS             f78a8000 31ec6c6d - vga.sys
f7760000 31ec6ccb - Msfs.SYS                 f7400000 31ec6cc7 - Npfs.SYS
fcdbe000 31eed262 - NDIS.SYS                 a0000000 31f954f7 - win32k.sys
fcd80000 31ee8583 - vga.dll                  fcdba000 31ec6e6c - TDI.SYS
fccfe000 31efe3f9 - nwlnkipx.sys             fcced000 31ed0750 - nwlnknb.sys
fcccc000 31f130a7 - tcpip.sys                fccb0000 31f50a65 - netbt.sys
f7808000 31ec6e04 - amdpcn.sys               f7640000 31f8f864 - afd.sys
f76c0000 31ec6e7a - netbios.sys              fcc4f000 325c3856 - srv.sys
fcc10000 31f5003b - rdr.sys                  fcbff000 31f7a1ba - mup.sys
f7470000 31ed0752 - nwlnkspx.sys

Address  dword dump   Build [1381]                                - Name
801470e8 fccdc4b5 fccdc4b5 00000002 00000246 00000246 fccb0da7 - tcpip.sys
801470f8 fccb0da7 fccb0da7 806984e8 806a6a28 80785690 8069e5dc - netbt.sys
80147110 fcce5208 fcce5208 00000000 80147178 00000000 fccd0023 - tcpip.sys
80147118 80147178 80147178 00000000 fccd0023 806a0023 00200000 - ntoskrnl.exe
80147120 fccd0023 fccd0023 806a0023 00200000 8014719c ffffffff - tcpip.sys
8014712c 8014719c 8014719c ffffffff 8062b3e8 ffffffff 00000030 - ntoskrnl.exe
8014714c 80147178 80147178 00000000 fccdc4b5 00000008 00010296 - ntoskrnl.exe
80147154 fccdc4b5 fccdc4b5 00000008 00010296 806a6acc 806a6a28 - tcpip.sys
80147174 801471a8 801471a8 801471a8 fccd3a72 00000000 8014719c - ntoskrnl.exe
80147178 801471a8 801471a8 fccd3a72 00000000 8014719c ffffffff - ntoskrnl.exe
8014717c fccd3a72 fccd3a72 00000000 8014719c ffffffff 806a6a2c - tcpip.sys
80147184 8014719c 8014719c ffffffff 806a6a2c 806a6a28 80782462 - ntoskrnl.exe
8014719c fccb0630 fccb0630 00000000 00000003 801471ec fccd3074 - netbt.sys
801471a8 801471ec 801471ec fccd3074 806a6a28 00003850 80147264 - ntoskrnl.exe
801471ac fccd3074 fccd3074 806a6a28 00003850 80147264 00000002 - tcpip.sys
801471b8 80147264 80147264 00000002 8076a8ac 8078244e 80782457 - ntoskrnl.exe
801471ec 80147230 80147230 fcccf10f 8076a8a8 926fa8c0 bd6fa8c0 - ntoskrnl.exe

Beginning dump of physical memory
```

Figure5.12 Target System

6. CONCLUSION

6.1 Summary of Work Done

The number of security-breaking attempts, originating inside the organizations is increasing steadily. Attacks made in this way, usually done by "authorized" users of the system, cannot be immediately located by existing DIDS. As the load increase in the network existing DIDS may not detect all the attacks in the network. The reason for the same is that there is a chance for packet loss at monitor station. Existing DIDS requires more bandwidth between hosts systems and monitor stations. In our proposed work, we have implemented novel IDS which address the problems of existing DIDS.

By replicating NIDS at secondary monitor station and providing synchronization between replicated monitor stations we addressed the problem of detection rate drop due to increase in load on the network. To trace the point of origin of an attack we used mobile agents along with a NIDS. By sending only required summaries between hosts and monitor stations we have minimized bandwidth requirement. We have tested our proposed system with the Ping of death, Teardrop, Winnuke and Syndrop attacks. We have showed that it successfully finds the origin of the attacks.

6.2 Suggestions for Further Work

The work in this dissertation can be extended as follows

- The proposed system can be extended to multi level Tracing system. In that monitor station creates one Tracing Agent for each attack that is found by NIDS. This Tracing agent is dispatched to the host from where the attack is generated. Tracing agent observes the log to check whether that attack is really generated by that system or remote station. If the attack is generated by remote station Tracing Agent is dispatched to that corresponding remote station to find point of origin of attack this process continue until Tracing find exact position.
- The proposed IDS can be combined with techniques like probabilistic packet marking and bloom filters to trace the attacks that are from outside the network.

REFERENCES

[1] Mukherjee, B.Heberlein, T.Levitt, “Network Intrusion Detection”, IEEE network, pages26-41, 1994.

[2] James P. Anderson Co., Fort Washington, “Computer Security Threat Monitoring and Surveillance” Technical report, pages 1-56, 1980.

[3] D. E. Denning, “An intrusion-detection model”, IEEE Transactions on Software Engineering”, Vol. SE-13(No. 2):pages 222-232, Feb. 1987.

[4] Karen Scarfone, Peter Mell, “Guide to Intrusion Detection and Prevention Systems”, National Institute of Standards Technology Special Publication on Intrusion Detection Systems 2001, SP 800-94, pages 1-124, Feb 2007.

[5] Peng Ning, Sushil Jajodia, “Intrusion Detection Systems Basics”, in Hossein Bidgoli (Ed), Handbook of Information Security, John Wiley & Sons, pages 685-700, 2005.

[6] Nita Patil, Chhaya Das, Shreya Patankar, Kshitija Pol, “Analysis of Distributed Intrusion Detection Systems using Mobile Agents” First International Conference on Emerging Trends in Engineering and Technology ICETET, pages 1255-1260,2008.

[7] Aaron Hector and V. Lakshmi Narasimhan, “A New Classification Scheme for Software Agents”, Proceedings of the Third International Conference on Information Technology and Applications, pages 1-6, 2005.

[8] Lange, Danny, and Mitsuru Oshima, “Mobile agents with Java: The Aglet API. World Wide Web, vol. 1, no. 3 , pages 111-121, 1998.

[9] G.A. Aderounmu, B.O. Oyatokun, “Remote Method Invocation and Mobil Agent: Remote Method Invocation and Mobil Agent”, Issues in Informing Science and Information Technology, Volume 3,pages 1-11, 2006.

[10] Abdulrahman Hijazi, Nidal Nasser, "Using Mobile Agents for Intrusion Detection in Wireless Ad Hoc Networks", Second IFIP International Conference on Wireless and Optical Communications Networks, pages 362 – 366, 2005.

[11] Pham V.A. and Karmouch A., "Mobile Software Agents: An Overview"; IEEE Communication Magazine, pp. 26-37,1998.

[12] J. S. Balasubramaniyam, J. O. Garcia-Fernandez, D. Isacoff, E. Spafford, and D. Zamboni. "An architecture for intrusion detection using autonomous agents". In Proceedings of the 14th Annual Computer Security Applications Conference, pages 13-24, 1998.

[13] Midori Asaka, Atsushi Taguchi, Shigeki Goto. "The Implementation of IDA: An Intrusion Detection Agent System". Technical report, pages 1-10, 1999.

[14] F. Barika, N. EL Kadhi, and K. Ghedira, "Intelligent and mobile agent for intrusion detection system: Ima-ids," Technical Report, pages 294-309, 2003.

[15] J. D. De Queiroz, L. F. R. Da Costa, and L. Pirmez. "Micael: An autonomous mobile agent system to protect new generation networked application". In 2nd Annual Workshop on Recent Advances in Intrusion Detection, pages 1-10, 1999.

[16] M. Benattou, and K. Tamine. "Intelligent Agents for Distributed Intrusion Detection System", Proceeding of World Academy of Science, Engineering and Technology, Volume 6 pages 190-194, 2005.

[17] "A signature of an attack", [Last Accessed: Feb 2009] http://www.symantec.com/business/security_response/attacksignatures/index.jsp.

[18] F. Cuppens and R. Ortalo. "LAMBDA: A Language to Model a Database for Detection of Attacks". Proceedings of the Third International Workshop on the Recent Advances in Intrusion Detection (RAID'2000), Toulouse, France, October 2000.

[19] Peter Jay Salzman, Michael Burian Ori Pomerantz "The Linux Kernel Module Programming Guide" ,Peter Jay Salzman, ver 2.6.4,2007.

[20] "An overview of inotify-java", http://www.den-4.com/page/31, [last accessed: March: 2009].

[21] "An overview of jpcap 0.7: a Java library for capturing and sending network packets", http://netresearch.ics.uci.edu/kfujii/jpcap/, [last accessed: Feb 2009].

Jpcap:

Jpcap[21] is an open source library for capturing and sending network packets from Java applications. It provides facilities to:

- capture raw packets live from the wire.
- save captured packets to an offline file, and read captured packets from an offline file.
- automatically identify packet types and generate corresponding Java objects (for Ethernet, IPv4, IPv6, ARP/RARP, TCP, UDP, and ICMPv4 packets).
- filter the packets according to user-specified rules before dispatching them to the application.
- send raw packets to the network.

In our work, for reading packets in the network we have used JpcapCaptor class from Jpcap open library. For capturing packets from a network, the first thing we have obtained list of various network interfaces on our machine. To do so, we have used getDeviceList( ) method. This getDeviceList( ) method is provided by Jpcap's JpcapCpator class. Once we have obtained the list of network interfaces and choose which network interface to capture packets from, we can open the network interface by using Jpcap's openDevice( ) method, which is also method from JpcapCaptor class.

```
NetworkInterface[] devices = JpcapCaptor.getDeviceList();
int index=...; // set index of the interface that you want to open.
//Open an interface with openDevice(NetworkInterface intrface, int snaplen, boolean
promics, int to_ms)
JpcapCaptor captor=JpcapCaptor.openDevice(device[index], 65535, true, 20);
```

When calling the JpcapCaptor.openDevice( ) method, we have specified the following parameters:

Name:	**Purpose**
NetworkInterface intrface	Network interface that you want to open.
int snaplen	Max number of bytes to capture at once.
boolean promics	True if you want to open the interface in promiscuous mode, and otherwise false. In promiscuous mode, you can capture packets every packet from the wire, i.e., even if its source or destination MAC address is not same as the MAC address of the interface you are opening.
int to_ms	Set a capture timeout value in milliseconds.

JpcapCaptor.openDevice( ) returns an instance of JpcapCaptor. Once we obtained an instance of JpcapCaptor, we captured packets from the interface. The PacketReceiver interface defines a receivePacket() method, so we have implemented a receivePacket() method in our work such that, it stored the packets that it captured in ArrayBlockingQueue. The following sample implement a receivePacket() method which simply prints out a captured packet.

```
class PacketPrinter implements PacketReceiver {
        //this method is called every time Jpcap captures a packet
 public void receivePacket(Packet packet) {
        //just print out a captured packet
  System.out.println(packet);
 }
 }
```