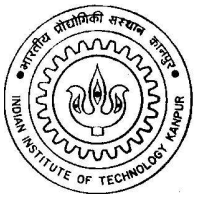
Intrusion Prevention and Automated Response in *Sachet* Intrusion Detection System

A Thesis Submitted
in Partial Fulfillment of the Requirements
for the Degree of
Master of Technology

by

Chinmay Niranjan Asarawala



to the

Department of Computer Science & Engineering
Indian Institute of Technology, Kanpur

May, 2006

Certificate

This is to certify that the work contained in the thesis entitled "*Intrusion Prevention and Automated Response in Sachet Intrusion Detection System*", by *Chinmay Niranjan Asarawala*, has been carried out under my supervision and that this work has not been submitted elsewhere for a degree.

May, 2006

(Dr. Dheeraj Sanghi)
Department of Computer Science & Engineering,
Indian Institute of Technology,
Kanpur.

Abstract

An Intrusion Detection System (IDS) monitors computer systems and network traffic, analyzes the traffic to identify possible security breaches, and raises alerts. It is difficult for human users to analyze the alerts and take swift appropriate actions which gives the attacker enough opportunity to continue to carry out further attacks. It is therefore important to take some automated actions to stop the attack. Unlike intrusion detection system, which passively monitors and reports, an Intrusion Prevention System (IPS) sits inline between the attacker and the system, monitors the traffic and stops the attacker to carry out attacks rather than just reporting them.

In this thesis, we describe the design and implementation of automated response module for Sachet - A distributed, real-time, network-based IDS developed at IIT Kanpur. The aim of automated response is to take immediate action in response to alerts generated by IDS to protect the system from further attacks. We are able to achieve a response time of less than one second.

We also describe the design and implementation of Intrusion Prevention System (which works independent of Sachet). Our intrusion prevention system detects signature-based attacks using INTEL IXP2400 Network Processor. It drops the packets containing predefined alert signature patterns thereby preventing these packets to reach the system. We tested our IPS on DARPA dataset and are able to achieve a speed of 24 Mbps without packet loss.

Acknowledgments

I take this opportunity to express my sincere thanks to my thesis supervisor, Dr. Dheeraj Sanghi, for his support and guidance. He provided me with many valuable ideas throughout the thesis period. I also thank Prabhu Goel Research Center and Army Technology Board for partially supporting my thesis. I would also like to thank my project partner, Palak Agarwal, for his co-operation and innovative suggestions. I would also like to thank my colleagues, Vinaya Natarajan, Devendar Bureddy, Satyam Sharma and Dungara Ram Choudhary for their suggestions. I would like to thank all my classmates too for making my stay at IITK enjoyable.

Finally, I would like to thank my parents and my beloved wife Payal for their constant support and encouragement in all my endeavours.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

E-business has become a powerful business model for companies where customers transact with the organization over the Internet rather than in person or through other communication medium/the phone. The business of many companies has spread across many countries. This has changed the traditional way of handling customers during specific hours of day. The companies need to keep their computer resources available twenty-four hours to the customers. Most of these resources are part of an enterprise computer network with important data. The losses incurred due to lack of availability of organization's resources to customers or theft of intellectual property amounts to billions of dollars every year. It is therefore important to protect an organization's information systems and networks from unauthorized access and intrusion.

1.1 Intrusion Detection System

An Intrusion Detection System (IDS) is a software/hardware tool used to detect unauthorized access to a computer system or a network[14]. It gathers information from various areas within a computer or a network and analyzes it to identify possible security breaches including intrusions (attacks from outside the organization) and misuse (attacks within organization). It tries to increase the availability, integrity, and confidentiality of computer systems, thereby providing security.

There are several ways to categorize an IDS depending on how it collects data, the methodology to analyze data and the action it takes. An IDS is classified broadly in two types depending on the way it collects data:

Host-based IDS Host-based IDS uses system and application logs as its data source. A Host-based IDS consists of an agent on a host which identifies intrusions by analyzing system calls, application logs, file-system modifications (binaries, password files, etc.), and other host activities.

Network IDS A Network IDS uses network traffic as its data source. In a Network IDS, sensors are placed at strategic points within the network to capture all network traffic flows and analyze the content of individual packets for malicious activities such as denial of service attacks, buffer overflow attacks, etc.

Each approach has its own strengths and weaknesses. Some of the attacks can be detected only by host-based or only by Network IDS. Some IP-based DOS (Denial of Service) attacks can only be detected by examining the headers of the packets. Host-based IDS do not see the packet header, so they cannot detect this type of attacks while Network IDS can detect them. Similarly, many switch-based encrypted network pose problems for Network IDS. It becomes difficult to identify location where Network IDS should be kept to completely cover the network with unencrypted data available to it. Host based IDS mostly see unencrypted data before it reaches an application.

An IDS can also be classified in two categories according to the method they use to detect attacks:

Misuse Detection In a Misuse Detection system, also known as signature-based system, well known attacks are represented by signatures. A signature is a pattern of activity which corresponds to the intrusion it represents. The IDS identifies intrusions by looking for these patterns in the data being analyzed. The accuracy of such a system depends on its signature database. Misuse Detection systems cannot detect novel attacks as well as slight variations of known attacks.

Anomaly Detection An anomaly-based IDS examines ongoing traffic, activity, transactions, or behavior for anomalies on networks or systems that may indicate attack. The underlying principle is the notion that attack behavior differs enough from normal user behavior that it can be detected by profiling the normal behaviour and comparing it with the current one. By creating baselines of normal behavior, anomaly-based IDS systems can observe when current behavior deviates statistically from the norm. This capability theoretically gives anomaly-based IDS ability to detect new attacks for which the signatures have not been created. The disadvantage of this approach is that there is no clear method for defining normal behavior. Also, a user may have been trying some new activity in a legitimate fashion. Therefore, such an IDS can report an intrusion, even when the activity is legitimate.

An IDS is also classified in one of the following two categories based on the action it takes:

Passive IDS In a passive system, the IDS detects a potential intrusion, logs the information and signals an alert for a human to take necessary actions.

Reactive IDS In a reactive system, the IDS not only logs the information related to a potential intrusion but also takes actions. Reactive IDS generates an automated response in several ways:

- **Session Sniping** In this kind of response, the IDS sends a TCP reset message to the attacker or both parties to reset the TCP connection. Since, many of the attacks make use of UDP or ICMP packets, this response may not be possible.
- **ICMP Messaging** In this kind of response, an ICMP error message is sent to the attacker specifying that the victim, the victim's network or the destination port is unreachable. The problem with this response is that many protocol stacks are not coded strictly conforming to RFCs and the ICMP message may be ignored.
- **Shunning** In this kind of response, the attacking host is denied access to the whole target network or the target host or some specific services on

target host. A firewall is configured dynamically to block the suspected attacking host.

Though both IDS and Firewall relate to network security, an IDS differs from a firewall in that a firewall looks out for intrusions in order to stop them from happening. An IDS suspects the intrusion once it has taken place and then may take some action. This highly limits the effectiveness of IDS over firewalls. The problems in using IDS are:

1. **False Positives and Negatives** Anomaly Detection generates a large number of false positives if not properly implemented and the Misuse Detection fails to recognize the attacks not present in its signature database.

2. **Overhead** Most of the IDS generate a large amount of reports and consume a good amount of processing power.

3. **Delayed Response** An IDS generates alerts and then leaves to the administrator to take necessary actions. Human response is quite slow and intruder might succeed in invading the system.

4. **Inability to Monitor at High Traffic Rates** An IDS cannot monitor the traffic at high transmission rates.

False positives and negatives can be handled only by implementing the intrusion detection engine correctly. Overhead can be minimized and high speed can be achieved if the IDS is removed from the hosts and implemented on a separate host with high processing power. Response can be made quick if the deep packet inspection capabilities of IDS can be combined with the packet blocking capabilities of a firewall giving rise to automated response and intrusion prevention. We have tried to address these issues in this thesis.

1.2 Intrusion Prevention System

An Intrusion Prevention System (IPS) is any hardware/software device that has the ability to detect known and/or unknown attacks and prevent the attack from being

successful. They are classified in following categories[9]:

Inline Network IDS The difference between a Network IDS and an inline Network IDS is shown in Figure 1.1. Network IDS generally has a stealth interface (without an IP address) to monitor the traffic, while inline NIDS works like a layer two bridge, sitting between the intranet that need to be protected and the rest of the network. All the packets have to go through the inline IDS, which inspect the packets for vulnerabilities. If a packet is found suspicious, it is dropped and logged. The system should be very reliable because of its inline nature.

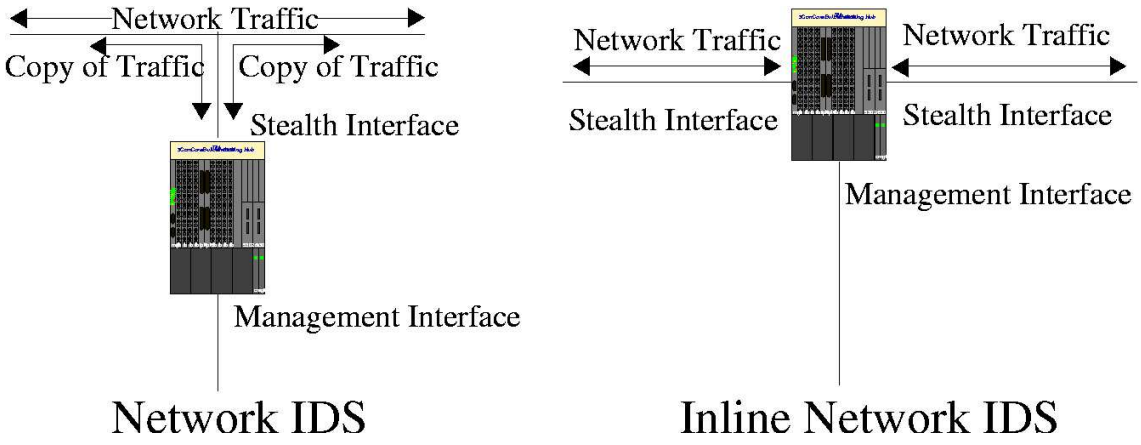


Figure 1.1: Difference between NIDS and Inline-NIDS

Layer Seven Switches They are generally used for load-balancing purpose, but also include DoS or DDoS protection. They generally are signature-based.

Application Firewalls They need to be installed on every host and protect specific applications by monitoring the interaction of user with the application and the interaction of application with the operating system. They are generally anomaly based. They generate a profile of the normal activity and monitor any deviation.

1.3 Related Work

There are various products available which implements automated response or can be classified as Intrusion Prevention Systems. We discuss some of them in this section.

1.3.1 Automated Response in IDS Products

- *CISCO Secure IDS*

CISCO Secure IDS[3] is a Network IDS that uses signature database to detect intrusions. The system consists of sensors that monitors network packets in real-time and a Director platform used to configure, log and display alarms generated by sensors. It implements following automated responses:

- TCP reset
- IP blocking (shunning)

- *Intrusion SecureNet Pro Sensor*

Intrusion SecureNet Pro Sensor[4] uses signature-based intrusion detection and uses TCP reset as an automated response. The sensor supports 5,000 signatures with 1,00,000 simultaneous connections.

1.3.2 Intrusion Prevention Systems

- *TippingPoint Intrusion Prevention System*

TippingPoint's IPS[13] provides protection through total packet inspection at gigabit speeds. It combines the power of Network Processor with ASICs to provide protection to application against cyber attacks and to infrastructure such as routers, switches against traffic anomalies.

- *Juniper Networks Intrusion Prevention Solutions*

Juniper Networks Intrusion Detection and Prevention(IDP)[5] uses misuse as well as anomaly detection to provide protection against current and emerging threats at both application and network layer with a centralized control. The IDP is deployed inline to provide protection against worms, Trojans, spyware, keyloggers and other malware.

- *CardGuard*

CardGuard[2] is a Network Intrusion Detection/Prevention system implemented on a single IXP1200 network processor. It scans reconstructed TCP streams and UDP packets for snort[10] signatures.

Our design of Intrusion Prevention System has been highly influenced by CardGuard.

1.4 Problem Statement and Approach

In this thesis, we describe the design and implementation of Automated Response module for Sachet - a distributed real-time network-based intrusion system with centralized control, developed at IIT Kanpur[17, 18, 19, 20]. We also describe the design and implementation of an Intrusion Prevention System on Intel IXP2400 that works independently of Sachet.

Our approach for Automated Response is as follows: We have integrated Sachet with a firewall to implement shunning (blocking of IP packets) as an automated response. We maintain a database of pre-configured responses for each signature-based attack. We call it *Sid Response Map*. Each signature is identified with a unique identifier called 'sid' (short form of signature id). The sid response is in terms of a seven-tuple:

- Interface
- Source IP Address

- Source Port
- Destination IP Address
- Destination Port
- Protocol
- Duration

As soon as an intrusion is detected, based on the configuration, a block request is executed at the firewall to deny access to the suspected attacking host. If any of the Source IP, Source Port, Destination IP, Destination Port or Protocol value is true in the configuration, the suspected packet's respective field is used in issuing the block request on the specified network interface. For example, if source IP address field is true, the packet's source IP address is used to block packets coming from it. This gives finer granularity to IDS and allows to block only some particular service or the complete host. After duration amount of time is elapsed, the block request is revoked.

The Intrusion Prevention System that we have developed implements signature-based intrusion detection on Intel IXP2400 Network Processor with Aho-Corasick[15] algorithm for pattern matching. Our approach is simple. Each packet coming through port 0 of the network card is examined for Snort signatures and if a packet matches any one of the signatures, it is forwarded on port 2, otherwise it is forwarded on port 1. Thus only packets without any known malicious attack can pass from port 0 to port 1. A packet sniffer utility can be run on a host to capture all the packets coming from port 2 and store them in database for analysis in future. We have tested our IPS with DARPA Dataset[1] and have been able to achieve a speed of up to 24 Mbps without any packet loss and up to 30 Mbps with only 1% packet loss. We have also developed and implemented a variation of Aho-Corasick algorithm and compared the performance.

1.5 Organization of Report

Chapter 2 presents the architecture of Sachet and its components. Chapter 3 presents the design and implementation of Automated Response module in Sachet. Chapter 4 presents the design and implementation of Intrusion Prevention System. Chapter 5 presents the results of testing the Automated Response Module and the Intrusion Prevention System. Chapter 6 presents conclusion and future work.

Chapter 2

Architecture of Sachet

In this chapter, we describe the architecture of *Sachet* Intrusion Detection System. Later sections describe the components of the system and the interaction between them. Detailed information can be found in [17, 18, 19, 20]. In Section 2.3, we describe the changes made to the Sachet architecture and the Sachet protocol for incorporating Automated Response Module.

The architecture of Sachet is as shown in Figure 2.1. Sachet components communicate with each other using the Sachet protocol. The protocol provides reliability, mutual authentication, confidentiality and integrity of all messages.

2.1 Sachet Modules

Sachet consists of several modules such as the Server, different kinds of Agents and the Console. We describe them in this section.

2.1.1 The Sachet Server

The Server provides centralized control for managing multiple Agents which are deployed at critical points of an enterprise network. It collects alerts from multiple agents and stores them in the database. It can run in the background as a daemon or service as well as a user process and is installed on a dedicated machine. The Server does not have any user interface and cannot directly interact with the user.

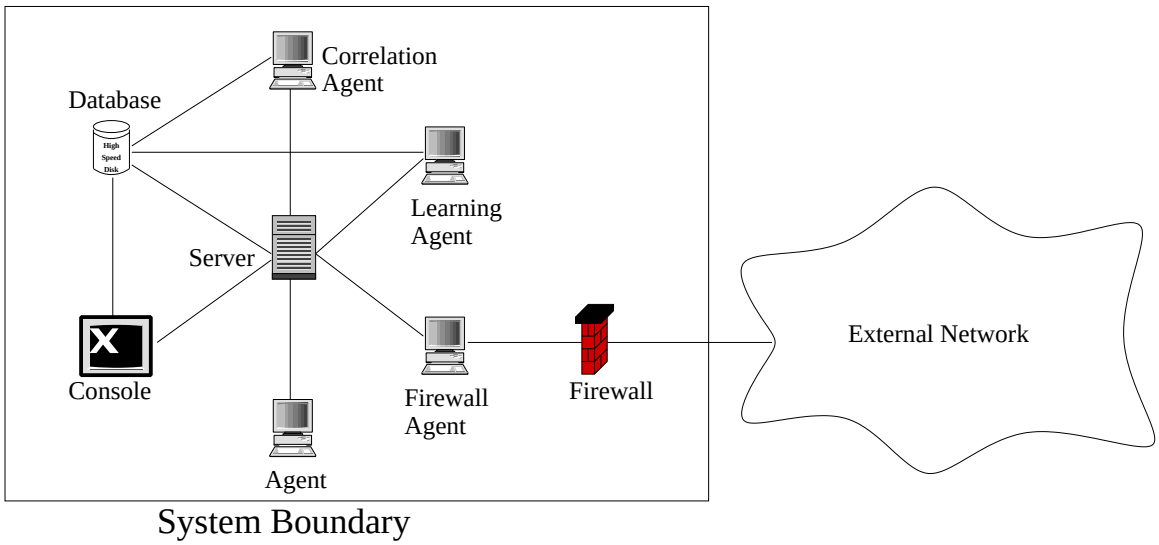


Figure 2.1: Architecture of Sachet IDS

The Console is used for controlling the Server. The Server communicates with the Console, which is a separate process, using a simple request-response protocol in which the Console sends a request for some information or a command to carry out a task and the Server responds by providing appropriate information or result. The Server periodically monitors the health of each Agent and reports it to the Console. It maintains information about Agents in a database and retrieves it at the beginning of its execution.

2.1.2 The Sachet Agent

The Agent is an application which can run in background and does not interact with the user. It comprises of four sub-components: Misuse Detector, Anomaly Detector, Vulnerability Assessment Module and Control Module. The control module creates separate processes to run Misuse Detector, Anomaly Detector and Vulnerability Assessment. The Control Module monitors the health of each of this process and reports it to the server. The Misuse Detector uses open source software

Snort[10]. The Anomaly Detector constantly compares the normal profile generated by Learning Agent with the network traffic it scans and reports anomalous behaviour if it finds any deviation. Vulnerability Assessment module periodically scans the system for vulnerabilities and generates a vulnerability profile. When Control Module receives alerts from the Anomaly Detector as well as from the Misuse Detector, it adds a priority to the alert based on the vulnerability profile and sends these alerts to the Server. If the system is vulnerable for the alert, the priority is kept high otherwise it is kept low. Misuse Detector and Anomaly Detector can be stopped/started/restarted as desired by the Control Module. Control Module also periodically monitors them and reports their status to the Server. References[17, 18] describe working of Sachet Agents in detail.

2.1.3 The Sachet Learning Agent

The Learning Agent constructs normal profile of the network by using features extracted from network traffic by Agents. Reservoir Sampling, a stream handling technique, and Support Vector Clustering, an unsupervised learning technique, are used for generating the normal profile of the system. The profile generated by the Learning Agent is used by Anomaly Detector in Agents for detecting deviations in the observed patterns of network activity. Reference[20] describes Learning Agent in detail.

2.1.4 The Sachet Correlation Agent

The Correlation Agent retrieves alerts from the database and correlates them using prerequisite-consequence based alert-correlation. The output are correlation graphs which shows alerts of a single sequence of attacks under a single node. Reference[19] describes Correlation Agent in detail.

2.1.5 The Sachet Console

The Sachet Console provides an interface to the administrator through which he can interact with the system. The administrator can configure, monitor and control the system from a central location. The administrator can add, modify or delete an Agent, enable and/or disable signatures of a particular agent, start/stop anomaly/misuse detector, and start/stop learning or correlation. It also displays the information stored in the database which are of use to the administrator. The Console provides a mechanism by which the signature database of misuse detector can be remotely updated for every Agent.

2.2 The Sachet Protocol

The Sachet Protocol is used for communication between the system components. It addresses issues of security, reliability and scalability. It provides graceful degradation in that the system doesn't break down completely if some of the components of the system fails.

2.2.1 Sachet Server-Agent Protocol

Sachet Server-Agent protocol is used for communication between the Server and the Agents (Agents, the Learning Agent, and the Correlation Agent). It uses UDP as the transport layer protocol. Sachet uses a public-key cryptography algorithm, RSA, for authentication between the Server and Agents. The Server and Agents know each others' authentic public keys. The authentication mechanism is based on challenge-response method and allows Agents and the Server to authenticate with each other and negotiate on symmetric cryptographic key before transmitting any application data. Once the symmetric cryptographic key is established, the whole communication takes place encrypted with this key.

The packet structure is as shown in Figure 2.2. The following describes each field in detail:

Encryption Type This field is used to indicate the method used for encrypting the

packet. It can take three different values which indicate that packet is either encrypted with public key or with symmetric key or not at all encrypted.

PacketID This field contains a number that identifies each unique packet sent or received and can be used for detecting duplicates.

Agent ID Each Agent is recognized by a fixed and unique number called agent ID. It contains the agent ID of the Agent which sent the packet. Agent ID value of the Server is zero so as to distinguish it from the Agents.

Data Length This field gives the length of data in bytes.

Message Type This field describes the type of message such as, an alert message, probe message, command message etc.

Data This field is interpreted based on the value of Message type. Data is encrypted with public key during authentication phase, and afterwards with the session key.

Hash This field contains the encrypted hash (MD5) for the entire packet. It provides packet integrity and ensures that the packet has not been modified or damaged in transit. The hash is encrypted with private key during the authentication phase and with session key afterwards. Here, session key refers to the shared secret key that is set up during the authentication phase.

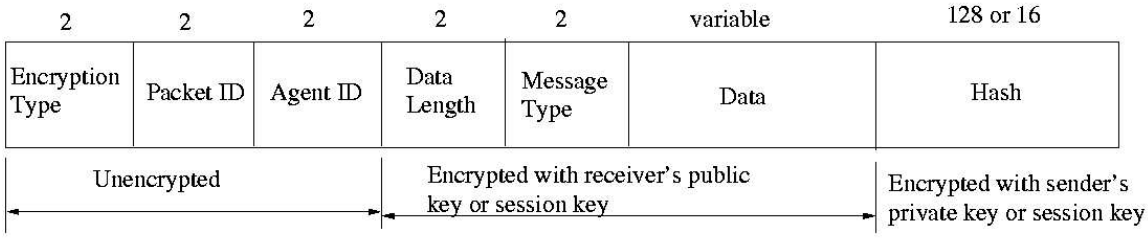


Figure 2.2: Packet Format for Sachet Server-Agent Protocol

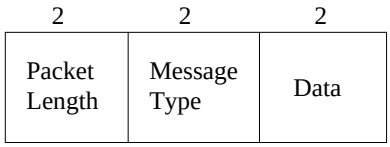


Figure 2.3: Packet Format for Sachet Server-Console Protocol

2.2.2 Sachet Server-Console protocol

The protocol is mainly designed for local communication between the Server and the Console. The Server and Console should be installed on the same host. The Console must authenticate itself to the Server with a password before issuing any instructions or requests. It uses TCP as a transport layer protocol. The Sachet Server-Console protocol packet format is as shown in Figure 2.3.

Packet Length It is the size of the complete packet in bytes.

Message type It indicates the type of packet. The packet can be a command message, a request-message, or a response-message.

Data This field contains data that is specific to the message type.

2.3 Incorporating Automated Response Module in the Sachet Architecture

In this section, we describe the changes made to the architecture and components of Sachet to incorporate Automated Response Module. Automated Response is carried out by a special Agent known as Firewall Agent. The major change to the architecture was the addition of Firewall Agent to Sachet.

Whenever any alert is detected, the Server retrieves the configured response from the configuration file and issues *block* requests to the Firewall Agent. The Server also maintains a timeout of block requests in the configured response. After the interval specified in the configuration file, the Server issues *unblock* requests to the Firewall Agent. Server stores these requests in the database.

The Firewall Agent needs to be started just as an Agent. It authenticates with the server like other Agents, and waits for a command from the Server. Firewall Agent has separate plugins for different firewalls. Firewall Agent at a time is configured to work with only one firewall. When the Firewall Agent starts, it reads the configuration and enables the plugin for that particular firewall. When the Firewall Agent receives any *block* or *unblock* requests from the server, it forwards them to the corresponding plugin and the plugin issues *block* and *unblock* requests to the firewall. Several new messages were added to the Sachet Protocol to incorporate Firewall Agent.

Console was modified so that it gives information about the Firewall Agent in a separate screen. The *block* requests issued by server and the pending *unblock* requests are displayed in this screen.

Chapter 3

Automated Response

Automated Response module is required to interact with several firewalls. Firewalls do not provide a unified interface to interact with them. In Section 3.1 we describe various options that we compared. In Section 3.2, we describe the implementation details regarding Automated Response module. In Section 3.3 we describe the changes applied to the current components.

3.1 Options Explored

We found various options for interacting with firewalls. We explore each one of them in detail:

Snortsam[11] Snortsam comes with an agent which interacts with several different firewalls and a plugin for Snort which, when alerts are detected, issues *block* and *unblock* requests to the agent. Directly integrating snortsam with the system had several advanatages and disadvantages:

- Advantages
  1. Less Development Time.
  2. Future enhancement in snortsam will help upgrade application without much efforts.
- Disadvantages

1. Snortsam doesn't use a database. If a block request has been issued and an application crashes, the site will remain blocked, leading to DoS (Denial of Service) attack.
2. Future versions of snortsam may change protocol of taking requests, countering the advantage 2.
3. It doesn't have authentication mechanism similar to Sachet. So it's difficult to use without any changes and one has to develop patches for it.
4. Application will be restricted with features provided by snortsam.

OPSEC[8] OPSEC is a security integration platform that lets developers access security components with a unified interface. Suspicious Activity Monitoring Protocol (SAMP) API allows Firewalls compliant to OPSEC to block the connection when an intrusion detection application identifies suspicious activity on the network or specified host. SAMP API defines an interface through which an intrusion detection application can communicate with a Management Server, which in turn directs the firewall modules to terminate the sessions or deny access to those specific hosts. Since Sachet was not developed with OPSEC in mind, integration of Sachet with OPSEC would have required a major effort and several changes to Sachet.

Because of the disadvantages offered by both the options, we decided to write the interface for firewalls ourselves.

3.2 Implementation

3.2.1 Sid Response Map

Sachet uses open source software Snort[10] to detect attacks. Latest version of Snort (version 2.4.4) has more than 4355 signatures. These signatures are classified in several categories based on the attack they signify. For example:

ICMP They include signatures of bad ICMP traffic or scanning tools.

Attack-Responses They include signatures of hosts that have already been compromised.

We have written automated response configuration for all these rules and call it Sid-Response Map. Most of the rules grouped together have similar kind of response. The automated response is specified in terms of 7-tuple with following fields:

Interface It specifies the interface on which the block request needs to be issued. Usually firewalls are installed on machines with two interfaces, one for intranet and one for external network. When we need to block a machine from intranet, we would like to give requests on intranet's interface, while blocking a system from external network would require the external interface. Its value can be 0 meaning internal interface or 1 meaning external interface.

Source IP It specifies whether the Source IP address field of the packet which generated the alert should be used for blocking, i.e. packets originating from Source IP of the suspicious packet be blocked. Its value can be True or False.

Destination IP It specifies whether the Destination IP address field of the packet which generated the alert should be used for blocking, i.e. packets destined for Destination IP of the suspicious packet be blocked. Its value can be True or False.

Source Port It specifies whether the Source Port field of the packet which generated the alert should be used for blocking. Its value can be True or False.

Destination Port It specifies whether the Destination Port field of the packet which generated the alert should be used for blocking. Its value can be True or False.

Protocol It specifies whether the Protocol field of the packet which generated the alert should be used for blocking. Its value can be True or False.

Duration It specifies for how long the suspicious host be blocked. It is in seconds. If its value is 0, the block is permanent.

Group	Sid	Iface	SrcIP	DstIP	SrcPort	DstPort	Proto	Duration
Responses	1292	Internal	True	False	False	False	False	0
Backdoor	103	External	True	False	False	False	False	0
Backdoor	107	Internal	False	True	False	False	False	0
Chat	541	Internal	True	False	False	False	False	60
Porn	1836	Internal	True	False	False	False	False	3600

Table 3.1: Examples of Sid-Response Map

Let's take the examples of Table 3.1. Sid 1292 has following signature

```
tcp $HOME_NET any -> $EXTERNAL_NET any (content:"Volume Serial Number")
```

Since this kind of packet is generated from a host which is compromised, the most intuitive response of the alert is to stop the host from accessing anything from external network. Most of the attacks involve downloading a file from some machines and then executing it. The automated response will be able to prevent this. Thus we want to block all the packets coming on *Internal* interface with *Source IP* equal to the Source IP of the suspicious packet, forever.

For sid 103, the signature looks like following:

```
tcp $EXTERNAL_NET 27374 -> $HOME_NET any (content:|0D 0A|[RPL]002|0D 0A|)
```

The signature is a part of backdoor group. The user is trying to attack the system, thus a simple response is to stop him by blocking packets coming from it. Thus we want to block all the packets coming on *External* interface with *Source IP* equal to the source IP of the suspicious packet, forever.

For sid 541, the signature is as follows:

```
tcp $HOME_NET any -> $EXTERNAL_NET any (content:"User-Agent|3A|ICQ")
```

The signature specifies that some employee of the organization is trying to use ICQ software for chatting purposes. If the policy of the organization doesn't permit chatting, we should punish the user by disabling his access to external net, say for 1 minute. That's what the response in Table 3.1 specifies.

The punishment for accessing restricted material from web should be more severe than chatting, and that's what the next signature specifies. Anybody on the internal network accessing pornographic content will be blocked access to external network for 3600 seconds, i.e., 1 hour.

In the design of the automated response module, Sid Response Map plays a very important role. A mistake in Sid Response Map can completely ruin the significance of automated response module. That's why it should be determined only with an expert's guidance.

3.2.2 Firewall Agent

Firewall Agent is a stripped-down version of an Agent, an Agent without the Misuse Detector, the Anomaly Detector and the Vulnerability Assessment module. It authenticates itself with the Server and waits for command from the Server. The Agent reads the configuration file at the start of the application and determines which firewall plugin needs to be enabled. *firewall_plugin* parameter in the configuration file specifies which plugin to be enabled. We have implemented only one plugin, for *iptables*, but new plugins can easily be added to the agent. Each plugin exports following functions:

- Init - It is called when the plugin is enabled. It doesn't take any parameters.
- Parse - It is called when the configuration file is being parsed, after a plugin has been enabled. It takes a key-value pair as parameters.
- Block - It is called when a block request is received from the server. It has six parameters namely Protocol, Source IP, Destination IP, Source Port, Destination Port, and Interface.
- Unblock - It takes the same parameters as Block.
- Exit - It is called when the application is to exit, with no parameters.

For plugin written for *iptables*, Init and Exit functions were empty because the firewall doesn't require any special initialization or exit. The Parse function would expect following parameters in the configuration files:

internal_interface - It specifies name of the internal interface e.g. eth0.

external_interface - It specifies name of the external interface e.g. eth1.

When Block function is called, it is translated into the following commands:

```
/sbin/iptables -i interface -s SourceIP -d DestinationIP -p Protocol
   --source-port SourcePort --destination-port DestinationPort
   -I FORWARD -j DROP
/sbin/iptables -i interface -s SourceIP -d DestinationIP -p Protocol
   --source-port SourcePort --destination-port DestinationPort
   -I INPUT -j DROP
```

Similarly when Unblock function is called, it is translated into the following commands:

```
/sbin/iptables -i interface -s SourceIP -d DestinationIP -p Protocol
   --source-port SourcePort --destination-port DestinationPort
   -D FORWARD -j DROP
/sbin/iptables -i interface -s SourceIP -d DestinationIP -p Protocol
   --source-port SourcePort --destination-port DestinationPort
   -D INPUT -j DROP
```

It is to be noted that some of the firewalls do not allow Destination and Source Port fields when the protocols are other than TCP and UDP. While writing Sid Response Map, this needs to be kept in mind if we want the application to run smoothly with all the firewalls.

3.3 Changes to Sachet

3.3.1 Sachet Protocol

Several new messages have been added to the Sachet Protocol to implement Automated Response Module in Sachet. They are described in Appendix B.

3.3.2 Sachet Server

The server identifies the Firewall Agent among all the Agents using the 'Type' field of the agents table in the database. Whenever an alert is detected, server determines the Source IP, Destination IP, Source Port, Destination Port, Protocol and Interface by combining suspicious packet with Sid-Response Map and sends a block request to Firewall Agent. It also stores the request in the database. If the Sid-Response Map has a timeout, the Server keeps an Unblock request pending in the database. After the specified duration elapses, the server issues the Unblock Requests to the Firewall Agent.

3.3.3 Sachet Console

The Console has been enhanced with a separate screen for Firewall Agent which shows already issued Block Requests and pending Unblock Requests.

Chapter 4

Intrusion Prevention

Our Intrusion Prevention System uses signature-based intrusion detection. Snort[10] has a standard set of signatures which were used as a signature database. We need to implement multi pattern matching for signature-based intrusion detection. In Section 4.1, we discuss various options that we looked at and why we decided on a particular algorithm. In Section 4.2, we explain the architecture of Intel IXP2400 and the components that affected our design. In Section 4.3, we briefly explain the algorithm for pattern matching with a small example and the changes we have made to suit our purpose. We also describe data structures used to store the pattern-matching information and other constraints of signatures. In Section 4.4, we describe a different approach of preprocessing the patterns and the benefit that we get from it. In Section 4.5, we describe the design and implementation of Intrusion Prevention System.

4.1 Pattern Matching Algorithms

We discuss below the advantages and disadvantages of several single as well as multi-pattern matching algorithms:

Aho-Corasick[15] It is the oldest multi-pattern matching algorithm. The algorithm creates a Finite State Machine (FSM) from the patterns. Each state in the FSM has a next state for every character. Some of the states have

output(s) that specify which patterns match when the state is reached. The algorithm then starts from the start state, takes one character at a time from input stream and determine the next state. If a state with an output is reached, the output of that state is reported. The creation of FSM takes $O(m)$ time and $O(m)$ space where m is the sum of the lengths of the pattern. If the next state can be obtained for any state-input pair in constant time, the scanning of pattern takes $O(n)$ time with constant space where n is the length of the input stream.

Boyer-Moore[16] It is a single-pattern matching algorithm. The algorithm creates two tables, one for each character in the input alphabet and one for each character in the pattern. Instead of scanning from left to right, the algorithm aligns the first characters of the pattern and the input stream and starts matching from the rightmost character of the pattern. Whenever any mismatch is found, the two tables are searched for the shift value corresponding to the input character and pattern character and the pattern is shifted right by the minimum of the two amount restarting the matching again at the last character. The algorithm takes $O(m+\alpha)$ time to compute the two tables and $O(m+\alpha)$ space to store them where m is the length of the pattern and α is the size of the input alphabet. The scanning takes $O(n/m)$ time in average case with constant space where n is the length of the input stream. For multi pattern matching, we will need to scan each pattern on the input stream. For small number of patterns the algorithm is simple to implement but for fairly large number of patterns, the algorithm performance deteriorates.

Wu-Manber[21] It is a multi-pattern matching algorithm based on Boyer-Moore[16] algorithm. It considers the pattern of minimum length and compares only first minimum length characters of each pattern. The algorithm requires manipulation of many data structures such as link list and hash tables. It is difficult to implement and manipulate these data structures on a network processor with very limited instruction set.

We found Aho-Corasick algorithm to be the most suitable and efficient algorithm for our purpose.

4.2 Architecture of Intel IXP2400

Figure 4.1 shows the block diagram of Intel IXP2400 Network Processor. Some of the components which affect our design are discussed below:

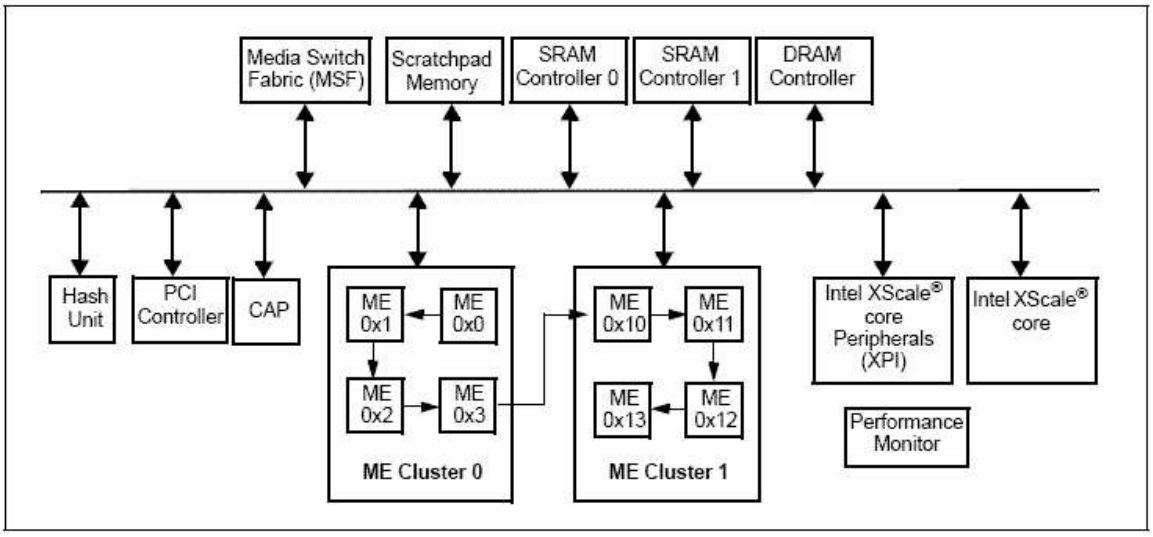


Figure 4.1: Intel IXP2400 Block Diagram

Microengines Intel IXP2400 has 8 microengines with each microengine having 8 threads and operates at 600 MHz. Each microengine also has 256 General Purpose Registers (GPR), 256 SRAM Data Transfer Registers, and 256 DRAM Data Transfer Registers. Data Transfer Registers are used to transfer data between SRAM/DRAM and the microengines. Each microengine has 4K 40-bit instruction store. All the threads share the same instructions but depending on the context they may execute different set of instructions. The threads are non-preemptive. The context switch takes about four cycles.

Memory Hierarchy Intel IXP2400 has 4 levels of memory hierarchy. Each has some special features. They are explained below in increasing order of access time:

Local Memory Each Microengine in Intel IXP2400 has 640 32-bit words of local memory. Typical access time for local memory is 3 cycles. It allows indexed addressing with post increment and decrement.

Scratch Memory Intel IXP2400 has a scratch memory of size 16K. It provides special bit operations. Typical access time of scratch memory is 60 cycles. It allows atomic operations and provide 16 rings (not used in our application).

SRAM Intel IXP2400 has two banks of SRAM, called SRAM0 and SRAM1, with a total size up to 128MB. The card that we have has 4 MB each, i.e. total 8MB. Typical access time is 90 cycles. Allows atomic operations and provide queue array with enqueue dequeue operations (not used in our application).

DRAM Intel IXP2400 can address up to 1GB of DRAM. Typical access time is 120 cycles. It has a direct path to and from Media Switch Fabric (MSF) which allows data to be moved between the two without going through processors. Thus packets are buffered in DRAM.

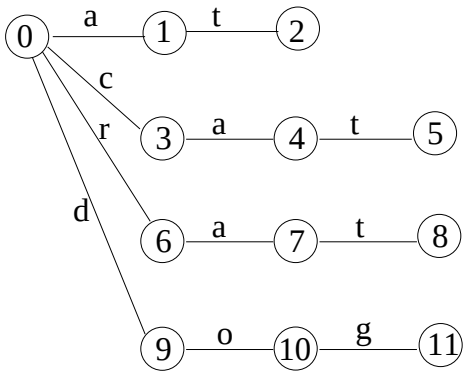
Except DRAM, all other memories are 4-byte aligned, i.e. read and write can be done only at 4-byte boundary. DRAM is 8-byte aligned and read-write operations happen at 8-byte boundary.

Ethernet Ports Intel IXP2400 has three 1-Gbps ethernet ports.

4.3 Data Structures

To describe the data structures used to represent pattern-matching information, we first show the working of the algorithm using an example. Figure 4.2 shows the FSM generated by running the preprocessing stage of the algorithm for the patterns {at, cat, rat, dog}.

A sample run on `saturation` is shown in Table 4.1. Since the scan visits the state 2 and 8 with outputs, the algorithm reports {at, rat} as the matched patterns.



State	Input	Next State
1,2,4,5,7,8,9,10,11	a	1
1,2,3,4,5,6,7,8,9,10,11	c	3
1,2,3,4,5,6,7,8,9,10,11	r	6
1,2,3,4,5,6,7,8,9,10,11	d	9

State	Outputs
2	{at}
5	{at,cat}
8	{at,rat}
11	{dog}

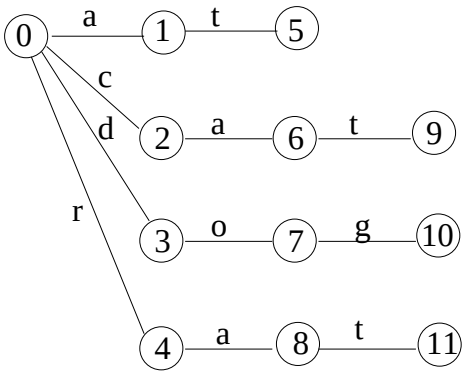
Figure 4.2: FSM Generated for patterns {at, cat, rat, dog}

Input	s	a	t	u	r	a	t	i	o	n
State	0	0	1	2	0	6	7	8	0	0	0

Table 4.1: Sample run of Aho-Corasick algorithm on "saturation" for FSM of Figure 4.2

Some of the observations from the algorithm:

- The algorithm generates the state numbers in Depth First Search (DFS) order.
- The depth of the FSM is equal to the longest pattern.
- Empirical evidence suggests that since the signatures are rare patterns, the algorithm will spend most of the time near the start state, i.e. only some small prefix of the pattern will match the input stream. Later we will show results to confirm this. To benefit from the locality, we renumbered the states in Breadth First Search(BFS) order to get the FSM shown in Figure 4.3. Thus if the top level states are kept in fast local memory, it becomes easy to determine whether a state is in local memory by simple comparison.



State	Input	Next State
1,3,5,6,7,8,9,10,11	a	1
1,2,3,4,5,6,7,8,9,10,11	c	2
1,2,3,4,5,6,7,8,9,10,11	d	3
1,2,3,4,5,6,7,8,9,10,11	r	4

State	Outputs
5	{at}
9	{at,cat}
10	{dog}
11	{at,rat}

Figure 4.3: FSM with states renumbered in Breadth First Search order

The most recent Snort signature database (version 2.4.4) has 4355 signatures. Running the preprocessing stage on these patterns resulted in an FSM with 15663 states. Since we want to find the next state for any state-input pair in constant time, the easiest way is to represent the FSM in a two dimensional matrix, where each row represents a state and column represents an input character. The structure of each state is as shown in Table 4.2:

Input State	0	...	a	c	d	g	o	r	t	...	FF	Out0	...	Out7	Outs
0	0	...	1	2	3	0	0	4	0	...	0	FFFF	...	FFFF	0
1	0	...	1	2	3	0	0	4	5	...	0	0	...	FFFF	1
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
11	0	...	1	2	3	0	0	4	0	...	0	1	...	FFFF	2

Table 4.2: Memory representation for FSM of Figure 4.3

We chose to have maximum eight outputs for each state which were found to be sufficient. Thus the total size of each state becomes 532 bytes, and with 15663 states, it takes around 7.9467 MB.

Besides the patterns in the payload of the packet, the signatures also specify constraints on the fields of the header. Only if these constraints are matched, the packet

contains an intrusion. Some of the constraints are quite complex and implementing them on Network Processor is difficult. So we chose a subset of the constraints and implemented them. The following explains each constraint in detail:

ip_proto It specifies the protocol field of the IP header. It can be UDP, TCP, ICMP or IP, e.g., ip_proto:6 means that the protocol must be TCP.

dsize It specifies the payload size. The constraint can be specified using '>', '<' and '=' operators, e.g., dsize:>325 means that the payload size must be greater than 325 bytes.

tos It specifies Type Of Service field in IP Header. It can be specified using '!' (not equal) or '=' operators, e.g., tos:!30 means that the Type of Service must not be 30.

id It specifies ID field in IP Header, e.g., id:333 means that the ID must be 333.

fragbits It specifies Flags field of IP Header. It can be R,M, or D with '+' or '*' symbols. The symbol '+' specifies the flags or more, '*' specifies any one of them, e.g., fragbits:MD* means either M or D flag must be set.

ttl It specifies TTL field of the IP Header. It can be specified using '>', '<', '=' or '-' (range) operators, e.g., ttl:23-25 means TTL must be between 23 and 25, both inclusive.

ports It specifies Source and Destination Port fields in TCP/UDP Header. They can be specified using '=', '!' (not equal) or '-' (range) operators. They are specified at the start of the signature specification in five tuple, e.g.,

```
tcp $EXTERNAL_NET 235 > $HOME_NET !21-56
```

means Source Port must be 235 and Destination Port must not be in [21,56].

seq It specifies Sequence Number field in TCP Header, e.g., seq:53233 means Sequence Number must be 53233.

ack It specifies Acknowledgement Number field in the TCP Header, e.g., ack:8888 means that Acknowledgement Number must be 8888.

flags It specifies Flags field in TCP Header. It can be F,S,R,P,A,U,1 or 2 with '+' or '*' symbols followed by the same flags. The first list specifies which flags are set and the second specifies which flags are to be ignored, e.g., flags:SA+,12 means that S,A or more flags must be set with 1,2 flags ignored.

window It specifies Window field of TCP Header. It can be specified using '!' (not equal) or '=' operator, e.g., window:8 means that Window field must be 8.

itype It specifies Type field of ICMP Header. It can be specified using '>', '<' or '=' operators, e.g., itype:8 specifies ICMP Echo Requests.

icode It specifies Code field of ICMP Header. It can be specified using '>', '<' or '=' operators, e.g., icode:<288 means that icode must be less than 288.

icmp_id It specifies ID field of ICMP Header, e.g., icmp_id:456 means that ID must be 456.

icmp_seq It specifies Sequence Number field of ICMP Header, e.g., icmp_seq:289 means that Sequence Number must be 289.

To store these constraints along with the FSM, we chose two more data structures shown in Table 4.3 and Table 4.4. The collection of the first we call Rule Mask Table and the second we call Rule Constraint Table. Together these two data structures store all the constraints for a specific sid.

Start(2)	End(2)	IPMask(20)	IPValue(20)	TCPMask(20) /UDPMask(20) /ICMPMask(20)	TCPValue(20) /UDPValue(20) /ICMPValue(20)

Table 4.3: Structure of each Rule Mask Table entry

The first data structure is maintained for each signature (identified with an sid). When a constraint asks for a specific value, say 'v', the Rule Mask Table is used.

Offset(2)	Size(1)	Opcode(1)	Value(4)

Opcode	Value
NOT EQUAL	0
GREATER THAN	1
LESS THAN	2
GREATER THAN OR EQUAL TO	3
LESS THAN OR EQUAL TO	4
AND	5

Table 4.4: Structure of each Rule Constraint Table entry

The mask of that field is set to all 1's and the value field is set to 'v'. The start and end specifies the start and end indices of other constraints for this sid in the Rule Constraint Table.

When a constraint specifies anything other than equality, the Rule Constraint Table is used. The offset field of the Rule Constraint Table entry specifies the offset of the field from the start of the IP Header, the size specifies the size of the field in bytes, the operator specifies one of the operator shown in Table 4.4 and the value specifies the value to compare with.

For example, take the following rule:

```
icmp $EXTERNAL_NET any > $HOME_NET any icode:>0 itype:18 sid:387
```

It specifies that for sid=387(hex 183) to match, the packet should be of ICMP protocol with ICMP Type field (Offset hex 0 in ICMP Header) = 18(hex 12) and ICMP Code field (Offset hex 15 from start of IP Header) > 0. The constraints will be represented using above data structures as shown in Table 4.5 and Table 4.6. In the Rule Mask Table, the Type field in ICMP (TCP) Mask is set to all 1's, i.e. 'FF' and the corresponding value field is set to '12'. There is only one entry for non equality comparison at index 5 for this sid, thus start and end indices equal 5. The entry at index 5 in the Rule Constraint Table specifies that field at offset 15 hex (ICMP Code) of size 1 byte should be greater than (opcode 1) 0.

Since the highest sid value is 5691, we keep 6000 entries in Rule Mask Table. With 84 bytes for each entry, it takes around 492 KB. The number of entries in Rule

Sid	Start(2)	End(2)	IPMask(20)	IPValue(20)	TCPMask(20)	TCPValue(20)
...	...	...	...	...	...	...
183	5	5	0	0	FF00...00	1200...00
...	...	...	...	...	...	...

Table 4.5: Sample Rule Mask

Index	Offset(2)	Size(1)	Opcode(1)	Value(4)
...	...	...	...	...
5	15	1	1	0
...	...	...	...	...

Table 4.6: Sample Rule Constraint

Constraint Table are not fixed so we allocate 64 KB for Rule Constraint Table. With 8 bytes for each entry we can store 8192 entries in the table which are sufficient.

It would generate false positives if we include signatures for which we haven't implemented all the constraints. This forced us to remove all signatures with such constraints and we were left with 1504 signatures which still represents a large chunk of attacks. The FSM generated for these signatures has 6172 states with 78 entries in Rule Constraint Table.

4.4 Different Approach

Next, we would like to describe a newer approach to the algorithm. The two-dimensional matrix that we use to store the generated FSM is sparse, in that many of the next state values lead to start state, i.e., state 0. This happens because for every character in the pattern, a new state is created which just has a single entry out of 256 inputs which leads to some next state and not to start state. In case of the complete snort signature database, the percentage utilization of the matrix is 48.9%. In case of the subset of signatures which we implemented, the percentage

utilization of the matrix is just 44.23%. Thus a lot of memory is wasted. To reduce the amount of memory wasted, we found a different way to create the FSM. We explain the approach below:

Byte Based Pattern	Bit Based Pattern
at	0110 0001 0111 0100
rat	0111 0010 0110 0001 0111 0100
cat	0110 0011 0110 0001 0111 0100
dog	0110 0100 0110 1111 0110 0111

Table 4.7: Patterns treated differently in two approaches

Instead of creating the next state for every input character (which is 1 byte in size), we chose to divide the byte in bits and use them to generate FSM. Thus instead of treating the patterns as a stream of bytes, we chose to treat them as a stream of bits. Let us revisit the sample we described in the previous section. Table 4.7 shows the difference between the patterns in the original Aho-Corasick algorithm and our bit based approach:

Thus every state will have just two next states, one for input=0 and one for input=1. It is to be noted that, the number of states will increase by 8 times on an average, so state number will take more space than in byte based approach. We chose to represent each state with 4 bytes rather than 2 bytes, making it 8 bytes per state. The scanning algorithm will also change. Instead of checking for a match at every input, we will have to check only at the end of every eighth bit.

The FSM generated using this approach for the sample described earlier has 75 states. Thus the amount of memory needed in byte based approach is 12 states * 512 bytes/state = 6144 bytes while in bit based approach it is just 75 states * 8 bytes/state = 600 bytes, a reduction of 90.23%. In case of the complete snort signature database, this approach generates 120,525 states which occupies, at 8 bytes per state, 964,200 bytes, which is 88% less than 8,019,456 bytes required for byte based approach. Similar results can be observed in case of the subset of signatures that we implemented. Bit-based approach generates an FSM with 47,196 states occupying 377,568 bytes while byte-based approach requires 6172 states * 512 bytes

= 3,160,064 bytes, a reduction of 88.06%. This reduction in memory utilization comes at a cost of performance hit as we show in Chapter 5.

4.5 Design of Intrusion Prevention System

So far we have described the preprocessing of signatures phase. In this section, we describe the signature-based intrusion detection phase. Table 4.8 shows the memory map that we have used to store the data structures:

Memory Type	Start Address	Size(Bytes)	Data Structure Stored
Local Memory	0	2560	Aho-Corasick FSM
SRAM0	66000	552K	Rule Mask Table
SRAM0	F0000	64K	Rule Constraint Table
SRAM0	100000	3M	Aho-Corasick FSM
SRAM1	40066000	3624K	Aho-Corasick FSM

Table 4.8: Memory Map of Data Structures

The application has three distinct tasks:

Receive a packet This involves getting the packet from the Media Switch Fabric (MSF), transfer it to DRAM and wake up one of the threads waiting to process a packet.

Process a packet This involves waiting for a packet to arrive. As soon as a packet arrives, its payload is scanned for the patterns. If a match is found, other constraints are checked and the packet is dropped (in our case sent to port 2) since a match indicates likelihood of intrusion. Otherwise it is sent to port 1.

Transmit a packet This involves getting the packet from DRAM buffer to MSF and then putting it in queue for transmission.

The receive and transmit phase requires physical device specific drivers which were readily available in a sample application provided along with the Montavista

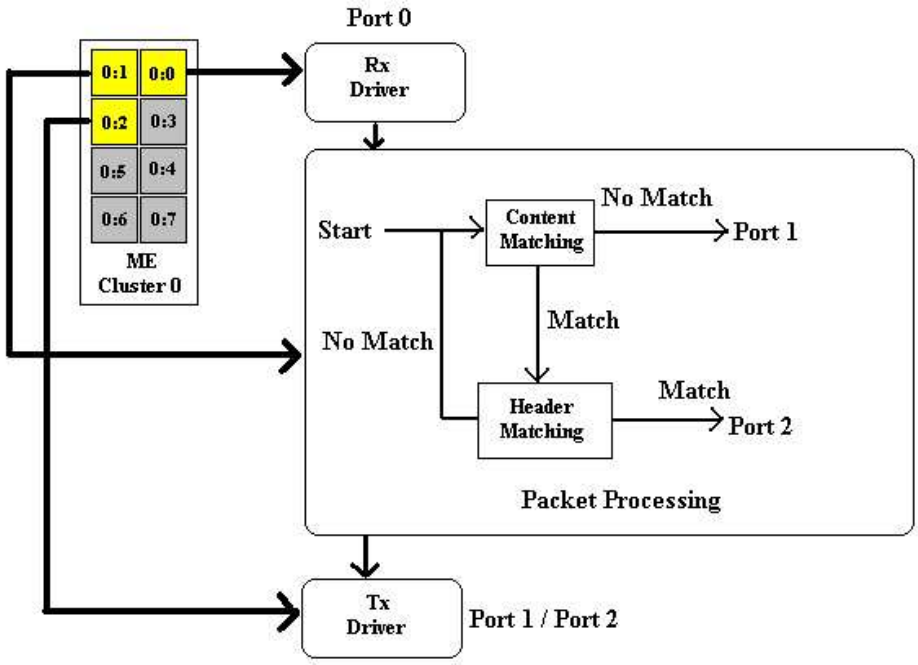


Figure 4.4: Intrusion Prevention System Architecture

Preview Kit received with the network card. The sample application was just counting the number of packets and total bytes received and was forwarding the packets in a circular fashion, i.e. packets from port 0 are forwarded to port 1, packets from port 1 to port 2 and so on. We reused the receive-transmit drivers, modified the processing part according to our application and mapped these three tasks on the microengines as shown in Figure 4.4.

Following pseudo code describes the working of packet processing task:

```
Content Matching
   Read the packet header in microengine registers
   Set the forwarding port to port 1
   Check for abnormal size packets, packets with IPV4 options or
      packets of protocol other than IPV4. If the packet belongs
```

```
      to any one of these categories than set forwarding port
      to port 2
   Initialize the current state to start state, i.e. state 0
   While the complete payload is not processed
      Take next character in the packet payload
      Find the next state from the current state and input character
      Set the current state to next state
      If the current state has outputs
         Call Match Header Constraints
         If Alert is detected
            stop processing the packet
         End If
      End If
   End While

Match Header Constraints

   For each output sid of the current state

      Retrieve the IP Mask, IP Value for the sid
      AND the IPMask with packet's IPHeader and compare with IPValue
      If does not match
         return
      End If
      Retrieve the TCP Mask, TCP Value for the sid
      AND the TCPMask with packet's TCPHeader and compare with TCPValue
      If does not match
         return
      End If
      Retrieve start and end indices of other constraints for the sid
      For every constraint from start to end in Rule Constraint Table
```

```
      Retrieve the offset, size, opcode and value for the constraint
      Retrieve size bytes from the packet header starting from offset
      Compare these bytes with the value according to the opcode
      If the constraint does not match
        return
      End If

    End For

    All the constraints are satisfied. Alert is detected.
    Set the forwarding port to port 2

End For
```

That finishes the design of our Intrusion Prevention System. Chapter 5 shows the result of testing it on the DARPA dataset.

Chapter 5

Experimental Results

5.1 Automated Response Module

5.1.1 The Setup

Four PCs were used to carry out testing on Automated Response Module. The Sachet Server was installed on the first PC, one Sachet Agent on the second and the Firewall Agent was installed on the third. The fourth PC (hostname "ids1") was used to carry out attack. All the PCs had RedHat Linux 9 operating System installed. The Firewall Agent was configured to use plugin for *iptables*. There were no entries in the *iptables* rules list when the test started.

5.1.2 The Test

A port scan utility nmap[7] executed on host "ids1" was used to carry out a port scan on the host where Agent was running. The time interval between the reporting of alerts to the Server and the *block* requests executed by the Firewall Agent was measured. The entries in the *iptables* rules list were also checked for the correctness. Nmap port scan typically generates two alerts from Sachet Agent, sid=469 and sid=1418. The response for both was configured in Sid Response Map as in Table 5.1.

Sid	Iface	SrcIP	DstIP	SrcPort	DstPort	Protocol	Duration
469	External	True	False	False	False	False	60
1418	External	True	False	False	False	False	60

Table 5.1: Test Sid Response Map

Chain	Target	Prot	opt	Source	Destination
INPUT(policy ACCEPT)					
	DROP	all	–	ids1	anywhere
FORWARD(policy ACCEPT)					
	DROP	all	–	ids1	anywhere
OUTPUT(policy ACCEPT)					

Table 5.2: Test Sid Response Map

5.1.3 The Results

The rules list of *iptables* had entries as shown in Table 5.2 after the nmap scan. The rules specify that all the packets coming from host "ids1" should be dropped, which was correct according to the configured Sid Response Map. The response time was less than one second. The Server issued *unblock* requests after one minute, as configured in Sid Response Map.

5.2 Intrusion Prevention System

5.2.1 The Setup

The Intrusion Prevention System was tested with two PCs, a Gigabit Ethernet switch and IXP2400 Network Processor. Intel IXP2400 Network Processor was mounted on PCI slot of one host and Minicom[6] was used to serially connect to the terminal of Network Processor. Tcpreplay[12] was used to replay the packets from the DARPA dataset dump file from the other host. The switch was configured to statically route the packets for Network Processor.

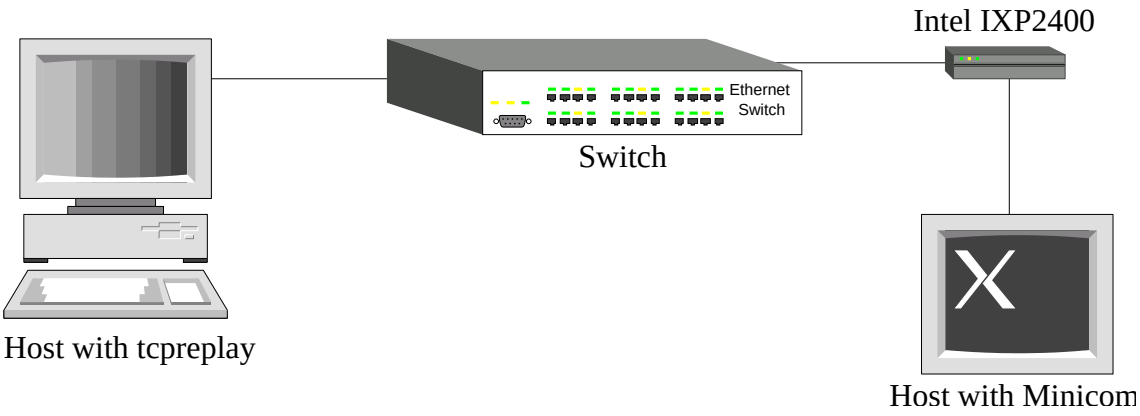


Figure 5.1: Setup for testing Intrusion Prevention System

5.2.2 The Test

Two set of counters were added to the IPS, one for counting how many times each state in the FSM is visited, and the other for counting how many times each level in the FSM is visited. We call them State counters and Level counters respectively. This counters will enable to confirm the assumption that the scanning algorithm spends most of the time in top level states. For carrying out the speed test, the DARPA dataset packets were replayed with increasing speed starting from 1 Mbps to 64 Mbps.

5.2.3 The Results

Figures 5.2, 5.3 and 5.4 prove our assumption that algorithm spends most of the time near the start state, i.e. state 0. States up to 113 make level 1 for the FSM which constitutes 83.76% of the accesses.

Figure 5.5 shows how the percentage of packets received deteriorates with the increasing speed rapidly after 32 Mbps. Till 30 Mbps the packet loss is less than 1%, but after that the IPS is not able to cope up with the traffic. With only the pattern matching, we are able to achieve almost double speed, upto around 58 Mbps with 2% packet loss. This suggests that on an average, equal amount of time is spent in matching patterns and other constraints. We found following reasons for this result:

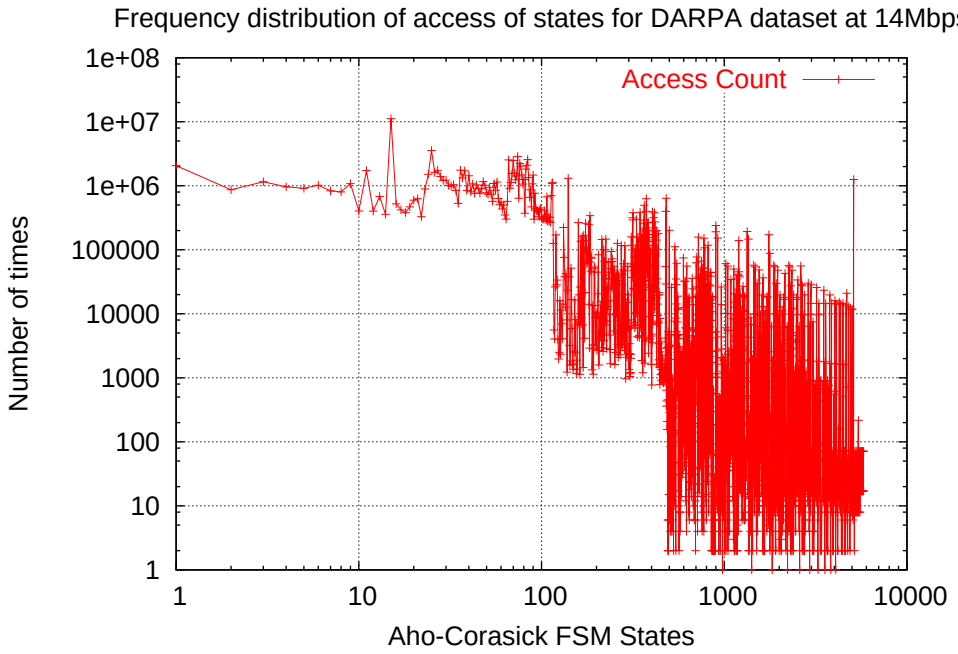


Figure 5.2: Frequency Distribution of Access to FSM States

1. Rule Mask Table is stored in SRAM. Reading IP Mask, Value and TCP Mask, Value requires accessing the SRAM 20 times.

2. Rule Constraint Table is stored in SRAM. Each rule requires one access to SRAM.

3. During the course of scanning one packet, if the pattern of the same signature matches again, the other constraints are still matched again. There are several small length patterns in snort signature database, that matches frequently with the packet content. Thus a lot of signatures falsely match first in pattern-matching and then discarded when other constraints do not match. This slows down the IPS.

The bit-based approach performs worse than the byte based approach. The plot shows that up to 12Mbps, the IPS is able to handle traffic, but after 14Mbps, the drop in packets increases rapidly. We analyzed this result as follows:

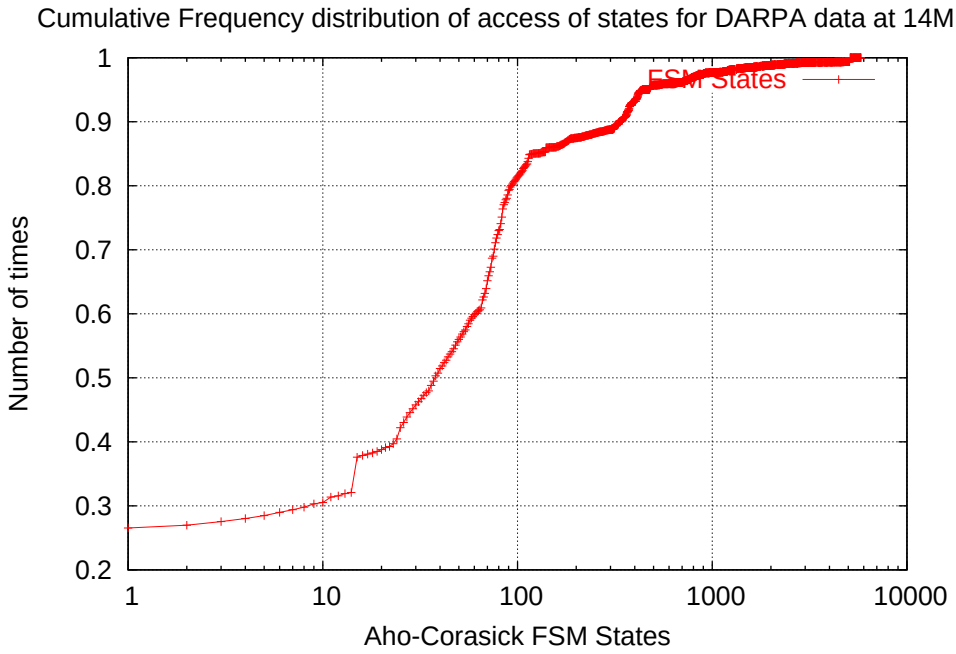


Figure 5.3: Cumulative Frequency Distribution of Access to FSM States

Local memory has access time of 3 cycles, while SRAM has access time of 90 cycles. Given the 2560 bytes of local memory, we are able to store just 4 states of FSM for normal algorithm and 320 states of FSM for bit-based algorithm in local memory. These states constitute 28.01% and 37.71% of access respectively. Assuming a simple latency model, the average access time for processing one input character is then

$$\begin{aligned} 0.2801 * 3 + (1 - 0.2801) * 90 &= 65.63 \text{ cycles, normal algorithm} \\ (0.3771 * 3 + (1 - 0.3771) * 90) * 8 &= 457.53 \text{ cycles, bit-based algorithm} \end{aligned}$$

The multiplication factor 8 in the second equation comes from the fact that we visit states 8 times in bit-based approach then in normal algorithm. Just to mention that we are neglecting the overhead involved in manipulating bits from byte. In general, if the states in local memory are accessed with a frequency of f, then the average

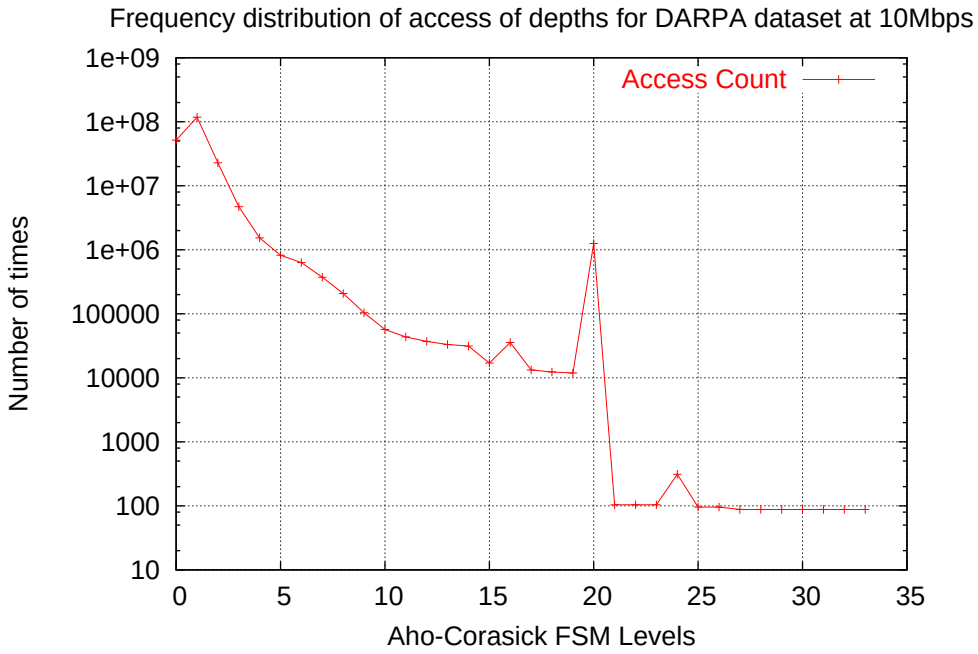


Figure 5.4: Frequency Distribution of Access to FSM Levels

delay in processing one input

$$f * l + (1 - f) * s \quad = \quad latency$$

where l is latency of access in local memory, s is latency of access in SRAM. For $f \in [0, 0.9]$, since $s >> l$,

$$(1 - f) * s \quad \approx \quad latency, \text{ for } f \in [0, 0.9], s >> l$$

If the above model holds true, the effect of increasing the memory size on the average latency will be as shown in Figure 5.6. The latency lines in the plot are purely made on the hypothetical model assumed above. They don't have any experimental evidence and we leave it for future work. We see that there is no performance gain obtained even when the local memory size is increased to around 40K.

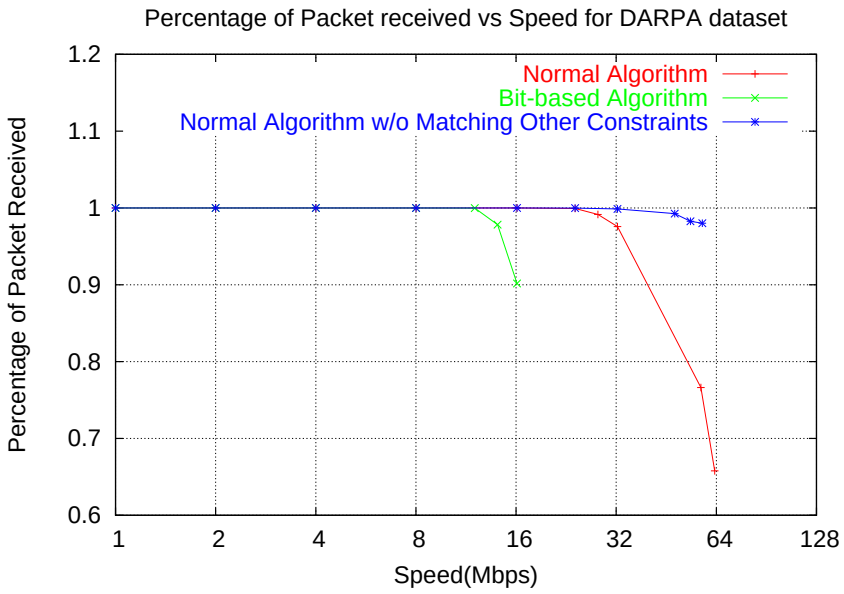


Figure 5.5: Intrusion Prevention System Speed Test

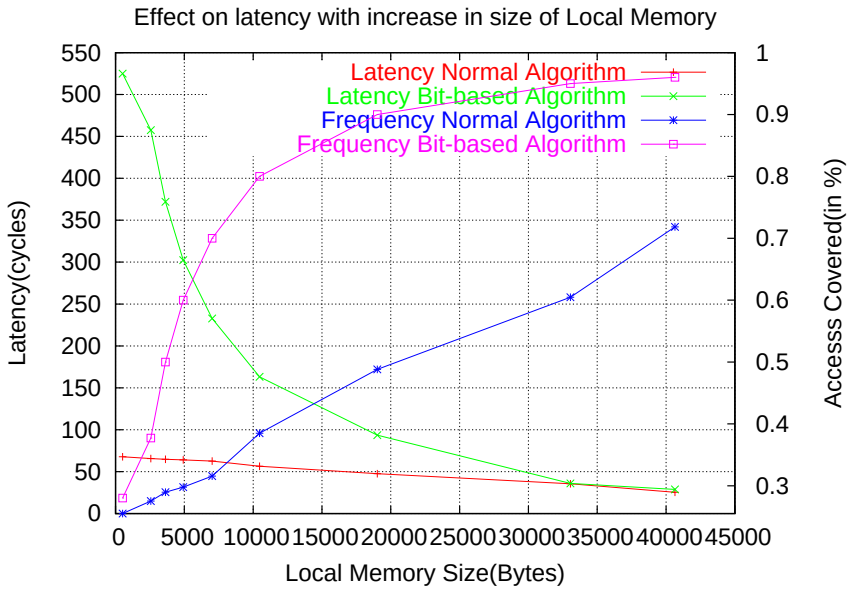


Figure 5.6: Expected effect on latency with increase in local memory size

Chapter 6

Conclusions and Future Work

We developed an Automated Response Module for *Sachet* Intrusion Detection System. We tested the module by generating alerts and checking for the block requests issued to the firewall. The system response time was less than 1 second. The module has some limitations. For example, it is vulnerable to DoS attack. An intruder may generate spoofed packets with Snort signature and the system will inadvertently block a legitimate website from being accessed from the internal network. Some of the products keep a *white* list of IP addresses which are never blocked. We have not explored the option and leave it as a future work.

We also developed a Network Processor based Intrusion Prevention System. We tested it with the DARPA dataset and were able to achieve a speed of 24 Mbps without packet loss. We implemented a variation of Aho-Corasick pattern matching algorithm and compared the performance. The IPS that we have developed has the following limitations:

1. Some signatures require more than one pattern to be present in the packet. We have not implemented it.

2. We didn't implement TCP Stream Reconstruction. So if a pattern spans multiple TCP packet, our IPS will not be able to recognize it.

3. We didn't implement all the constraints specified in Snort signatures. Some examples are, byte_test, byte_jump, pcre, distance, within, rawbytes, depth,

isdataat. See Snort documentation[10] for more information about them.

4. We haven't implemented case-insensitivity in pattern matching.

5. Signatures without any pattern are not matched.

The DoS limitation of Automated Response Module is handled by IPS.

References

[1] 1998 DARPA Intrusion Detection Evaluation. *http://www.ll.mit.edu/SST/ideval/docs/docs_ index.html*.

[2] Cardguard. *http://www.cs.vu.nl/~herbertb/projects/ffpf/*.

[3] Cisco Secure IDS. *http://www.ciscopress.com/articles/article.asp?p=24696&rl=1*.

[4] Intrusion SecureNet Pro Sensor. *http://www.intrusion.com/products/securenet/sensor/*.

[5] Juniper Networks Intrusion Prevention Solutions. *http://www.juniper.net/products/intrusion/*.

[6] Minicom, friendly serial communication program. *http://www.die.net/doc/linux/man/man1/minicom.1.html*.

[7] Nmap, Free Security Scanner for Network Exploration and Security Audits. *http://www.insecure.org/nmap*.

[8] OPSEC, Open Platform for Security.

[9] SecurityFocus. *http://www.securityfocus.com/infocus/1670*.

[10] Snort, open source network intrusion detection system. *http://www.snort.org*.

[11] Snortsam, an open source software consisting of a plugin for snort and an intelligent agent to interact with firewall. *http://www.snortsam.net*.

[12] tcpreplay, Pcap editing and replay tools for *NIX. *http://tcpreplay.synfin.net/trac/*.

[13] Tippingpoint Intrusion Prevention Systems. *http://www.tippingpoint.com/products_ips.html*.

[14] Wikipedia. *http://en.wikipedia.org/wiki/Intrusion_detection_system*.

[15] Aho, A. V., and Corasick, M. J. Efficient String Matching: An Aid to Bibliographic Search. *Communications of the ACM 18*, 6 (1975), 333–340.

[16] Boyer, R. S., and Moore, J. S. A fast string searching algorithm. *Commun. ACM 20*, 10 (1977), 762–772.

[17] Goel, S. Sachet - A Distributed Real-time Network Based Intrusion Detection System. Master's thesis, Indian Institute of Technology, Kanpur, June 2004.

[18] Jain, B. Intrusion Prevention and Vulnerability Assessment in Sachet Intrusion Detection System. Master's thesis, Indian Institute of Technology, Kanpur, June 2005.

[19] Kaur, P. Attack Scenario Construction and Automated Report Generation in Sachet Intrusion Detection System. Master's thesis, Indian Institute of Technology, Kanpur, June 2005.

[20] Murthy, J. V. R. Design and Implementation of an Anomaly Detection Scheme in Sachet Intrusion Detection System. Master's thesis, Indian Institute of Technology, Kanpur, June 2004.

[21] Wu, S., and Manber, U. A Fast Algorithm for Multi-Pattern Searching. *Technical Report TR-94-17* (1993).

Appendix A

Response Map

Some sample responses for Snort rules are given in this section. For the signatures, refer to Snort documentation[10].

Group	Sid	Iface	SrcIP	DstIP	SrcPort	DstPort	Proto	Duration
Attack Responses	1292	Internal	True	False	False	False	False	0
Backdoor	103	External	True	False	True	False	False	0
Backdoor	107	Internal	False	True	False	True	False	0
Bad-Traffic	524	External	True	False	False	False	False	600
Chat	541	Internal	True	False	False	False	False	60
Chat	540	Internal	False	True	False	True	False	60
DDOS	221	External	True	False	False	False	False	0
DNS	255	External	True	False	False	False	False	0
DNS	253	External	True	False	True	False	False	0
DOS	272	External	True	False	False	False	False	0
Exploit	1324	External	True	False	False	False	False	0
Finger	320	External	True	False	False	False	False	0
FTP	2546	External	True	False	False	False	False	0
ICMP-info	363	External	True	False	False	False	False	60

Group	Sid	Iface	SrcIP	DstIP	SrcPort	DstPort	Proto	Duration
ICMP	465	External	True	False	False	False	False	60
IMAP	1993	External	True	False	False	False	False	0
Info	488	External	True	False	False	False	False	60
Misc	500	External	True	False	False	False	False	0
Mutlimedia	1437	External	True	False	True	False	False	60
Mysql	1775	External	True	False	False	False	False	0
Netbios	537	External	True	False	False	False	False	0
Oracle	1673	External	True	False	False	False	False	0
Other-IDS	1760	Internal	False	True	False	False	False	0
P2P	549	Internal	False	True	False	False	False	600
Policy	553	External	True	False	False	False	False	600
POP2	1934	External	True	False	False	False	False	60
POP3	2121	External	True	False	False	False	False	60
Porn	1836	External	True	False	False	False	False	3600
RPC	601	External	True	False	False	False	False	0
Scan	613	External	True	False	False	False	False	1800
Shellcode	647	External	True	False	False	False	False	0
SMTP	654	External	True	False	False	False	False	0
SNMP	1893	External	True	False	False	False	False	0
SQL	676	External	True	False	False	False	False	0
Telnet	1430	External	True	False	False	False	False	0
TFTP	1941	External	True	False	False	False	False	0
Virus	721	Internal	False	True	False	False	False	0
Web-cgi	803	External	True	False	False	False	False	0
Web-client	1233	Internal	False	True	False	False	False	0
Web-coldfusion	903	External	True	False	False	False	False	0
Web-frontpage	1248	External	True	False	False	False	False	0

Group	Sid	Iface	SrcIP	DstIP	SrcPort	DstPort	Proto	Duration
Web-IIS	1970	External	True	False	False	False	False	0
Web-misc	2657	External	True	False	False	False	False	0
Web-php	1774	External	True	False	False	False	False	0
X11	1225	External	True	False	False	False	False	0

Appendix B

New Messages Included in the Sachet Protocol

The new messages added to the Sachet protocol for implementing Automated Response in Sachet IDS are described in this appendix along with their format. The packet format for these messages is shown in Figures 2.2 and 2.3. Here, we present only the format of the data part of these messages.

BLOCK The Server issues a *block* request to Firewall Agent using this message. The data part of this message contains a *BlockRequest* structure shown in Table B.1. The Firewall Agent replies using BLOCK_REPLY message code with a reply code BLOCK_OK or BLOCK_FAILED in data part indicating whether the request succeeded or failed.

UNBLOCK The Server issues an *unblock* request to Firewall Agent using this message. The data part of this message contains a *BlockRequest* structure shown in Table B.1. The Firewall Agent replies using UNBLOCK_REPLY message code with a reply code UNBLOCK_OK or UNBLOCK_FAILED in data part indicating whether the request succeeded or failed.

Field	Size(bytes)
Source IP	4
Destination IP	4
Source Port	2
Destination Port	2
Protocol	4
Interface	1
Duration	4

Table B.1: