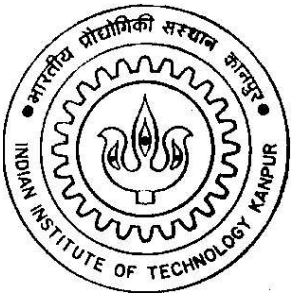
TCP Stream Reassembly and Web based GUI for Sachet IDS

by

Palak Agarwal



Department of Computer Science & Engineering

Indian Institute of Technology, Kanpur

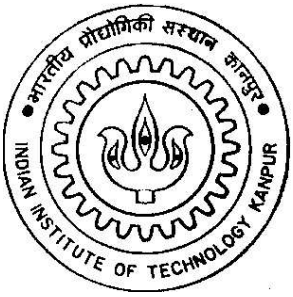
February 2007

TCP Stream Reassembly and Web based GUI for Sachet IDS

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

by

Palak Agarwal

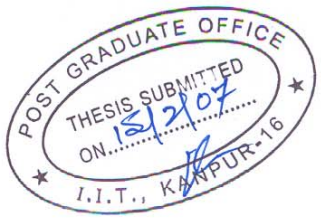


to the

Department of Computer Science & Engineering

Indian Institute of Technology, Kanpur

February 2007



Certificate

This is to certify that the work contained in the thesis entitled *"TCP Stream Reassembly and Web based GUI for Sachet IDS"*, by *Palak Agarwal,* has been carried out under my supervision and that this work has not been submitted elsewhere for a degree.

February, 2007

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

Abstract

Sachet is a Network based Intrusion Detection System developed at IIT Kanpur. It monitors the network traffic to detect any unwanted attempts to compromise the security of the network by malicious users. Recently an Intrusion Prevention functionality was also added to it. IPS monitors network traffic inline and prevent intrusions by dropping the malicious packets before they reach the actual host. In this thesis, we are adding two functionalities needed to enhance the utility of the system.

One of major technique to prevent an IDS/IPS from detecting an attack is through splitting the signature into two packets of a TCP Connection. As the IDS/IPS checks for signature in each packet individually it would not be able to detect this attack. However, on the host machine these packets would be reassembled and a stream of data is available to the application. Hence, it would get compromised. We are adding a TCP Reassembly module to our IPS so that it can detect those attacks which would have went though undetected.

We are also adding a Web based Graphical User Interface to the system so that a network administrator can monitor the IDS from a remote machine. Currently, the information about the alerts (and the status of network and nodes) can be monitored only from the server machine.

Acknowledgments

I take this opportunity to express my sincere gratitude towards my thesis supervisor, Dr. Dheeraj Sanghi, for his continuous 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 all the faculty members of Department of Computer Science for imparting invaluable knowledge on me.

I would also like to thank my project partner, Chinmay Asarawala, 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 for their constant support and encouragement in all my endeavours.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

With online business more important now than in yesteryears, importance of securing data present on the systems accessible from the Internet is also increasing. If a system is compromised for even a small time, it could lead to huge losses to the organization. Everyday new tools and techiniques are devised to stop these malicious attempts to access or corrupt data.

Traditionally firewall have been used to stop the intrusion attempts by an attacker. But firewalls have static configurations that block attacks based on source and destination ports and IP addresses. These are not sufficient to provide security from all the attacks. Therefore, we need IDS and IPS type systems which could analyse the payload of the packet to detect these attacks.

1.1 Intrusion Detection System

An Intrusion Detection System is a defense mechanism to monitor any unauthorized access to a system or a network. If an attack is detected, it is reported to the network administrator so that appropiate actions could be taken to provide security to the

system or network.

An IDS can be implemented in hardware as well as software. Software based IDS generally provide higher configurability but are slow. On the other hand hardware based IDS are difficult to configure but can handle higher network speeds. Also it is difficult to update hardware based IDS for new signatures. IDS can be categorized in a variety of ways depending on how they gather data, their methods to detect intrusion, and their architecture.

On the basis of collection of data, IDS are categorised into Network Intrusion Detection Systems (NIDS) or Host Intursion Detection System (HIDS). NIDS gathers data from the network traffic. It detects intrusions by analyzing the headers and the data present in the packets. It can reside anywhere on the network where it can read all the incoming packets to the set of hosts which it is monitoring. HIDS resides on the specific host which it is monitoring. It detects intrusions by analyzing the state of the host machine, information stored in RAM or disk, e.g. system call traces, event logs, etc.

On the basis of method to detect the attack, IDS can be categorized into Signature-detection based IDS and Anomally-detection based IDS. A signature based IDS uses a set of patterns (signatures) which are prepared manually from the logs of the earlier sucessful attacks. They work by looking for these signatures in the contents of the packets. Hence, they are very accurate and efficient. But these signatures need to be updated regularly to detect new attacks. An anomaly based IDS detects attacks on the basis of deviation from the normal activity. They use machine learning techniques to learn the normal behavior of the system/network over a long period of time. Any deviation from that behavior is considered an anomaly or an attack. Although this allows us to detect even unknown attacks but it raises a lot of false alarms. If the number of false alarms is too much, the actual alerts may get ignored.

On the basis their architecture, IDS are categorised into centralised and

distributed. In centralised IDS data is collected at different sources but is sent to a central server where it can be analysed for attacts. These IDS have a single point of failure. In distributed IDS, data is collected and analysed at various points in the network. The results are then sent to a central server to be displayed to the administrator and/or stored in a database.

1.2 Intrusion Prevention System

Intrusion Prevention System is a tool which not only looks for attacks on the system or network but also tries to stop them actively. They are considered an extension of Intrusion Detection Systems. They can also be network based or host based.

Network based IPS sits inline the network traffic flows and prevents attacks in real-time. Unlike traditional firewalls which do access control depending on the IP address and network ports, it also analyses the payload of the packet to detect intrusions. It can then take pre-emptive actions like dropping the network packet, resetting the session or blocking all traffic from that particular host. It should not have any false positives as that may lead to loss of legitimate traffic thus effecting the overall network performance. Host based IPS is installed on the system to be protected. It detect and block attacks by monitoring the actions performed by application on the system. For example, if a word processing application tries to access the system password then it will be marked as a malicious activity and would be blocked.

1.3 Problem Statement and Approach

Sachet is a software based Network Intrusion Detection System developed at IIT Kanpur [9, 10, 12]. Recently, Intrusion Prevention [5] functionality was also added to Sachet to actively prevent intrusion attempts rather than just showing alerts to

the administrator. Sachet IPS is developed on an Intel IXP2400 network card.

Sachet IPS sits inline the network traffic coming to the hosts. It receives individual packets from the network and performs signature detction on its payload. In a TCP connection, data is sent in form of multiple packets. These packets are sent independent of each other but are reassembled at the recieving end. This is transparent to the application running at receivers' end which treats it as a stream of data and not separate packets. Hence, splitting data into multiple packets creates scope for an attack as described below: When IPS scans a single packet for a signature it is possible that the signature is not present in that individual packet. However, the signature is spread across multiple packets belonging to the same connection. For instance, to detect an attack using PHF script vulnerability on Apache server, the string which is searched in the payload looks like "GET /cgi-bin/phf?". Now consider a packet with content ".....GET /cgi" and the next packet with content "-bin/phf.....". In this case, the IPS would not detect the signature in any of the packets and hence forward them to the host. On the other hand, host would reassemble the packets and get compromised as a result of the attack.

In this thesis, we design and implement a TCP Stream Reassembly Module to work with the IPS component of Sachet. Our approach for TCP Stream Reassembly is described as follows: When IPS receives a TCP packet, it is classified into the corresponding TCP connection to which this packet belongs based on the four tuple:

- Source IP Address
- Source Port
- Destination IP Address
- Destination Port

After that we determine whether it is actually the next expected packet of that connection by checking its TCP sequence number. If the packet is out of order

it is stored in a buffer. When the expected packet arrives, it is sent for signature detection. If the packet is clean it is forwarded to the intended host. However, if it matches a signature the packet is dropped. Also, all other packets belonging to the same stream are removed from the buffer.

There was one more problem with Sachet IDS that the Console which provides a Graphical User Interface to the Sachet IDS can run only on the machine on which server was running. So a network administrator can not view the alerts generated from any other machine in the network. We have developed a Web based GUI for Sachet IDS. This allows a network administrator to view the alerts generated by the agents from any machine with a web browser. It is developed in PHP and runs on Apache server.

1.4 Related Work

TCP Reassembly is an essential component of every Network Intrusion Detection or Prevention System. Some examples of IDS or IPS that support TCP Reassembly are:

- Snort [3] - Snort is an open software based Network Intrusion Detection System. It uses a software preprocessor to perform IP Defragmentation and TCP Reassembly before actual signature detection.
- CERCS Network Processor based IDS/IPS [8] - Center for Experimental Research in Computer Systems at Georgia Institute of Technology have developed a hardware IDS. It does IP Defragmentation and TCP Reassembly on an IXP 1200 network processor and actual Signature Detection on an Xilinx Virtex2 FPGA. The idea is to provide high end NIC cards with IDS.
- Cardguard [7] - Cardguard is a hardware NIDS developed on a IXP network card. It does TCP reconstruction and signature detection on the network

processor.

Our design of TCP Reassembly module is highly influenced by the Cardguard. We have added a garbage collection to avoid Denial of Service Attacks by an attacker.

1.5 Organization of the Report

The rest of this report is organized as follows. Chapter 2 describes the architecture of Sachet. In Chapter 3, techniques used by attackers to evade IDS/IPS from detecting attacks are explained. In Chapter 4 we discuss the architecture of TCP Reassembly module and changes made to Sachet IPS. Chapter 5 describes the design and implementation of web based GUI for Sachet IDS. In Chapter 6 we conclude and discuss the future work.

Chapter 2

Sachet

In this chapter we will give a brief description of the Sachet Architecture. Sachet has a software based IDS component and a network processor based IPS component. For a detailed description please refer to [9, 12].

2.1 Sachet Intrusion Detection System

Figure 2.1 shows the software components of Sachet IDS - Sachet agents, Sachet Server, Sachet Console, Sachet Learning Agent, Sachet Correlation Agent and Sachet Firewall Agent. The Sachet agent further comprises of four components - misuse detector, anomaly detector, vulnerability assessment module and the control module. Sachet system can be distributed over any number of hosts or sub-networks in a network. An agent monitors a host or a network segment for attack events in the network traffic that is incoming to the host or a network segment. The misuse detector and anomaly detector analyze network packets for patterns of attacks and generate alerts, and forward them to the control module. The control module authenticates with the server and subsequently sends all the generated alerts to the server over a secure and encrypted communication channel. The control module

starts and controls the misuse detector. It periodically monitors health of both the misuse detector and anomaly detector and takes appropriate action if any of the components fail. The server aggregates alerts from multiple agents and stores them in a log in a database. The server oversees the working of the agents and controls them by issuing commands to them. It also accepts requests and instructions from the console. The console is a GUI provided to user to configure, control and manage Sachet. The console provides powerful display capability to view alert information and detailed information of each agent. The console also provides capability of updating signature database for misuse detection and then communicating it to all agents. A console has to authenticate to the server before establishing communication with it.

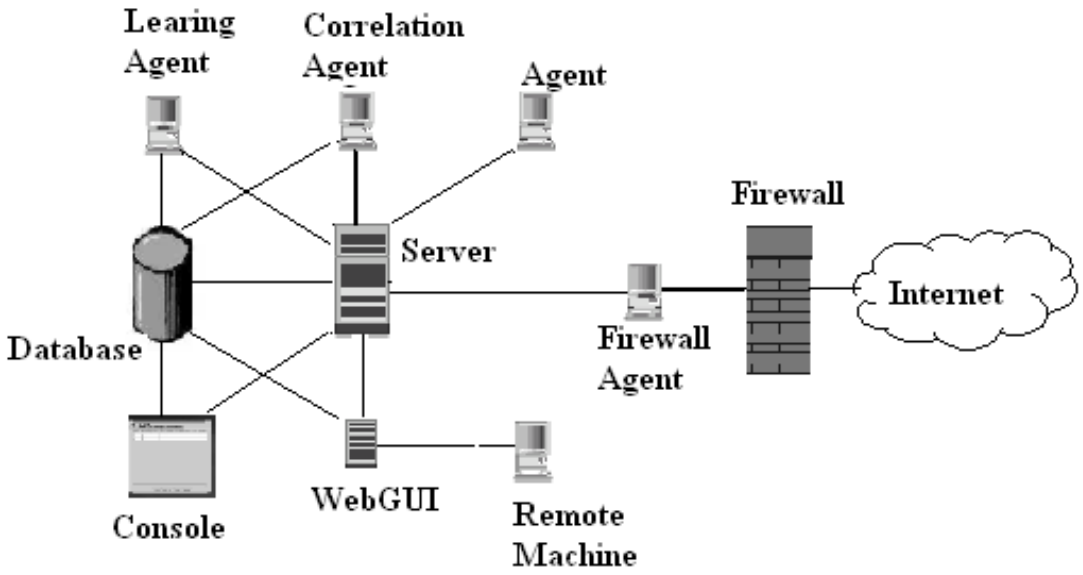


Figure 2.1: Architecture of Sachet

2.1.1 Sachet Server

The server is a central command authority of the Sachet IDS. It usually runs in background as a daemon or service and is installed on a dedicated machine. It is the centre of the Intrusion Detection System that oversees the functioning of all agents including learning, correlation and firewall agents. It authenticates agents and issues necessary commands to them whenever required. It also aggregates alerts from the agents which are then stored in a central database. It does not have its own user interface and communicates with the user through the GUI Console. 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 and the server responds by providing appropriate information or result. The user needs to authenticate to server by providing a pass-phrase before it can issue any commands to the server through the interface. 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 Sachet Agent

The Sachet agent passively monitors either the entire network traffic on a LAN segment, or an independent host. Any malicious activity is reported to the server as an alert over a secure channel. It can run in background as a daemon or a service and does not interact with the user. All commands are issued to the agents by the server. It has to authenticate itself before it can communicate with the server. After authentication, all alerts generated by it are sent to the server. The agent has four sub-components: Misuse detector, Anomaly Detector, Vulnerability Assessment module and Control module. Each of these sub-components run as separate processes on the target host.

The misuse detector uses an open source software called Snort [3]. Snort monitors IP packets, searches them for pre-defined patterns or signatures and gen-

erates corresponding alerts. The misuse detector then sends those alerts to control module which sends them to server for logging in the database. The signature used by the misuse detector needs to be updated regularly to protect the system from new vulnerabilities.

The anomaly detector analyses the network traffic and compares it with the normal profile generated by the learning agent. If it finds any aberrations from the normal profile then it is reported as an alert.

Vulnerability Assessment module uses an open source software called Nessus [2] for vulnerability assessment. Nessus is run periodically on the monitored machines and the detected vulnerabilities are stored in a file. This vulnerability information is used by the control module to mark the severity of the alert depending on whether the machine is actually vulnerable to the alert generated.

The Control module controls the functioning of the agent. It can start or stop the misuse detector and anomaly detector on instruction from the server. It receives alerts from them and sends them to the server. It also periodically monitors their health and reports it to the server.

2.1.3 Sachet Learning Agent

Learning Agent is a special agent which is used to generate a profile of the network which can be used by the anomaly detector. It uses reservoir sampling [15] to store the required data from the continuous flow of network data. It then uses Support Vector Clustering [6] technique to learn the features extracted from the network traffic. The network profile thus generated is then sent to the server whenever demanded. The server then stores this profile in the database and also sends it to the agents for updation.

2.1.4 Sachet Correlation Agent

Correlation Agent correlates the alerts generated by the misuse detector to give the network administrator a condensed high level view of the attack. It does incident reconstruction by matching the prerequisites of one attack with the consequences of an earlier alert on the same node. The related attacks are correlated into scenario and displayed on the console as a graph. Details about Alert Correlation in Sachet can be found in [11].

2.1.5 Sachet Firewall Agent

Firewall Agent receives block and unblock requests from the server and issues them to the firewall. It has separate plugins for different firewall. It enables the appropriate plugin at starting time after reading the configuration file. More detail about firewall agent is available in [5].

2.1.6 Sachet Console

The console provides a Graphical User Interface to the system administrator for controlling the entire system. It is a Java based application that runs on the same machine as the server. It provides a view of the information stored in the database whenever required. It is used to add, modify and delete agents to the system. It has separate screens for learning, correlation and firewall agents. It also allows the administrator to instruct the server to start and stop learning and correlation agents on specified client nodes. It allows the administrator to view the alerts generated from the agents and make decision to secure the system.

2.2 Sachet Intrusion Prevention System

Sachet IPS is built on an Intel IXP2400 network processor. It looks for specific patterns (signatures) in network packets for detecting attacks. Signature matching involves first searching a pattern in the payload of the packet and then constraint matching on the header fields to check whether the matched pattern is an actual attack or just a random match. Aho-Corasick Algorithm [4] is used for pattern matching.

2.2.1 IXP2400 Network Card

It has an Intel XScale core and 8 micro-engines. Figure 2.2 shows the block diagram of IXP2400 components. Each micro-engine has a clock of 600 MHz and have 8 hardware assisted threads. Each micro-engine has duplicate registers and program counters for each thread and can perform a context switch from one thread to the other quickly. The instrusion set of the microengines is especially tuned for network applications. The instruction set does not have any division or floating point operations. There is an independent instruction store of 4K instructions for each microengine. Each microengine also has a CRC unit to calculate CRC checksums.

Registers Each micorengine has four types of registers, namely, General Purpose Registers (256), SRAM Transfer registers (256), DRAM Transfer Registers (256) and Next Neighbour Registers (128). Transfer Registers are used to receive and transmit data from SRAM and DRAM. Next Neighbour registers can be set by the microengine to which they belong but can be read only by the next microengine in the sequence. Next Neighbour Registers can also be used as General Purpose registers but there is a delay of 16 clock cycles between data write and the next read. All the registers are divided equally between all the threads. But we can also have some or all registers in absolute mode which can be accessed by all threads within the microengine. This mechanism

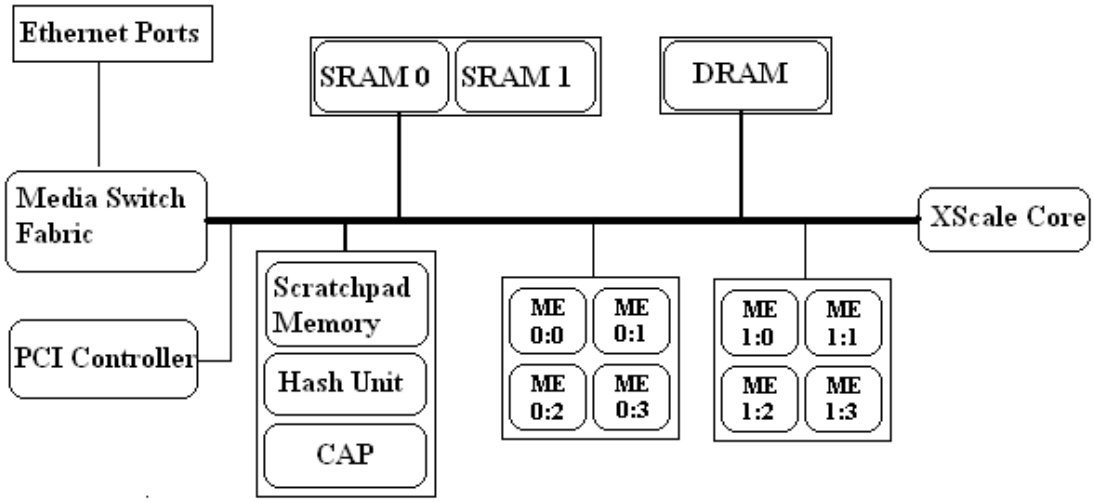


Figure 2.2: Intel IXP2400 Block Diagram

can be used to share information among the threads.

Memory IXP2400 has four different memory types with each having special features of its own.

Each microengine has a 2560 bytes of Local Memory. The local memory is shared by all the threads running on that microengine but not be accessed from any other microengine. Access time for each memory access is three cycles.

IXP2400 has a Scratchpad Memory of 16K which can be accessed by any microengine. The access time is approximately 60 clock cycles. It provides atomic bit operations.

It has two SRAM controllers with each of them supporting upto 64MB address space. Although our card has 4MB for each controller. It also provides atomic bit operations.

It also one DRAM controller that supports 1GB DRAM. Our card has a 128MB DDR RAM. DRAM has a unique functionality that it can move data

directly from and to Media Switch Fabric (MSF) without data going through the microengines. It is used as the buffer to store received packets.

Ethernet Ports IXP2400 has three 1-Gbps Ethernet ports.

2.2.2 Architecture of Sachet IPS

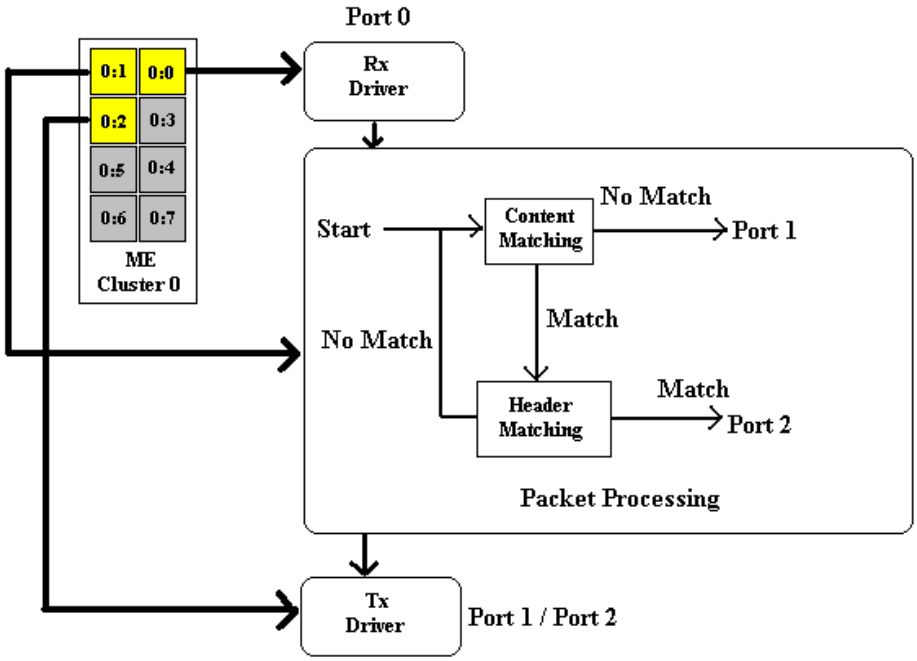


Figure 2.3: Intrusion Prevention System Architecture [5]

The sachet IPS has three main components: Receive, Process, and Transmit. Each component runs on a separate microengine on IXP2400. Figure 2.3 shows the maaping of each component to microengines.

- Receive module reveives network packets on port 0 and stores them in DRAM.

- Process module does the actual Signature Detection. All the packets that does not match a signature are marked for forwarding to port 1 and those which match a signature for port 2.
- Transmit module takes packets from DRAM and transmits them to appropiate port.

2.2.3 Aho-Corasick Algorithm

The algorithm uses a finite state machine to match multiple strings at a time. The FSM is constructed offline before the actual pattern matching is performed. For each state in the FSM there is a next state for every input character in the alphabet. Some of these states are output states that represent a pattern match. The important feature of FSM is that each state represents the required information about the input parsed till now, so we do not need to buffer the input. In our design, when a packet belonging to a particular stream is matched through Aho-Corasick algorithm we just have to save the final state of the FSM. For the next packet of the same stream we use that stored state as the start state of the FSM. Hence, we do not need to keep inorder packets in buffer to perform pattern matching on the stream. We can just forward the inorder packets and delete them from memory after an individual packet is analysed.

2.2.4 Data Structures and Memory Usage

FSM was represnted as a two dimensional matrix, where each row represents a state and column represent an input character. This design lets us find the next state in constant time as it is essentially just a memory lookup. Each state can have upto eight outputs which represent a matched pattern. The total size to store each state is 532 bytes. As we have 6172 states, it takes around 3.13 MB.

Besides Pattern matching Signature Detection also involves matching constraints on header fields. These constraints are matched only when a pattern is matched in the packet payload. The constraints are stored in two more data structures called Rule Mask Table and Rule Constraint Table.

Rule Mask Table is used when a constraint involves exact match of some fields to a value. It stores a mask for those fields and corresponding values to be matched. It also stores start and end indices of other constraints in Rule Constraint Table. There is a single entry for each signature in Rule Mask Table. Each entry is of 84 bytes. As we have less than 6000 entries, it takes around 492 KB.

Rule Constraint Table is used for any constraint other than equality. It has four fields:

- Offset field species the offset from the start of the IP header
- Size specifies size of field in bytes
- Opcode specifies the test operation
- Value specifies the value to compare with

Each entry takes 8 bytes. As the number of total entries is less than 8192, this table needs a maximum memory of 64 KB.

Chapter 3

IDS Evasion Techniques

In this chapter, we are describing some of the common methods employed by attackers to avoid detection by the IDS/IPS. Most of them were discussed by Ptacek and Newsham [13] way back in 1998. It was emphasised that the IDS/IPS cannot be effective as their view of network traffic is not identical to that of the application on the host.

3.1 Split Signature Attack

The split signature attack makes use of the fact that in a TCP connection, data is sent in form of multiple packets. On the other hand, an application running on the host machine is transparent to this. It treats it as a stream of data and not separate packets. An attacker can form packets in such a way that the signature is spread over multiple packets of the same connection. For instance, to detect an attack which uses PHF script on Apache server, the string which is searched in the payload looks like "GET /cgi-bin/phf?". Now consider if an attacker forms four packets with the following contents: ".....GET /", "cgi-", "bin/" and "phf.....". The IDS/IPS which sits inline will see these as invividual packets and would not

detect the signature in any of the packets and hence forward them to the host. The host would reassemble the packets and get compromised as the result of the attack.

This problem is accentuated because it is not necessary for the packets to come in same order in which they were sent. These packets are reordered by the receiver depending on the sequence number present in the packet. The sequence number is chosen randomly when the connection is initiated. It gets incremented by the number of bytes in the payload for every subsequent packet. Figure 3.1 shows an example of out of order packets that could lead to evasion by the IDS/IPS.

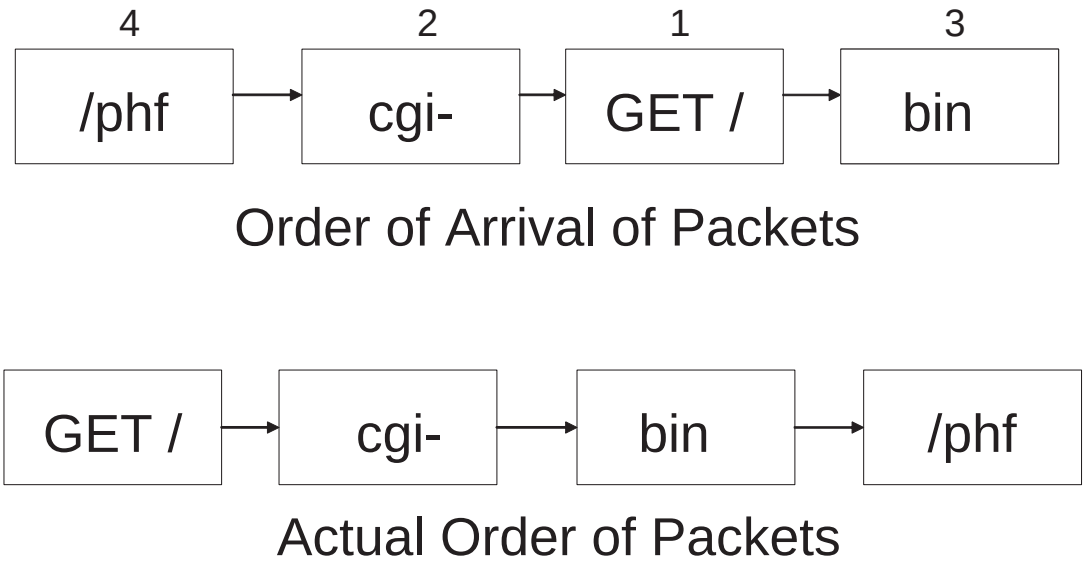


Figure 3.1: Reordering of Packets

The packets need to be reordered before signature match can be performed. Any IDS/IPS which does not reorder the packets would not have the same view as the host being protected. We are tackling this problem for Sachet IPS in this thesis.

3.2 Overlapping and Inconsistent IP Fragment Attack

This attack includes spreading signature over multiple overlapping IP fragments. The complexity of the problem is increased as different operating systems deal with overlapped fragments in different ways. Here are the different policies used to deal with overlapped fragments [14]:

- BSD - It keeps the content of that packet which has the lowest offset. A packet with an overlap is trimmed from the left till the point for which we have already received the data.
- BSD-right(HP JetDirect) - It keeps the content of the packet with the highest offset. Overlapped packets are trimmed from the right.
- Linux - It keeps the content of that packet which has the lowest offset. The new packet having same offset will overwrite the existing one.
- First(Windows) - Always accept the first value received for each offset in the packet.
- Last(Cisco IOS) - Always accept the last value received for each offset in the packet.

Figure 3.2 shows how overlapping and inconsistent IP packets would be received by operating systems with different policies.

Due to the this lack of consistency it is impossible for an IDS/IPS to detect the attack without having the exact knowledge of the internal network.

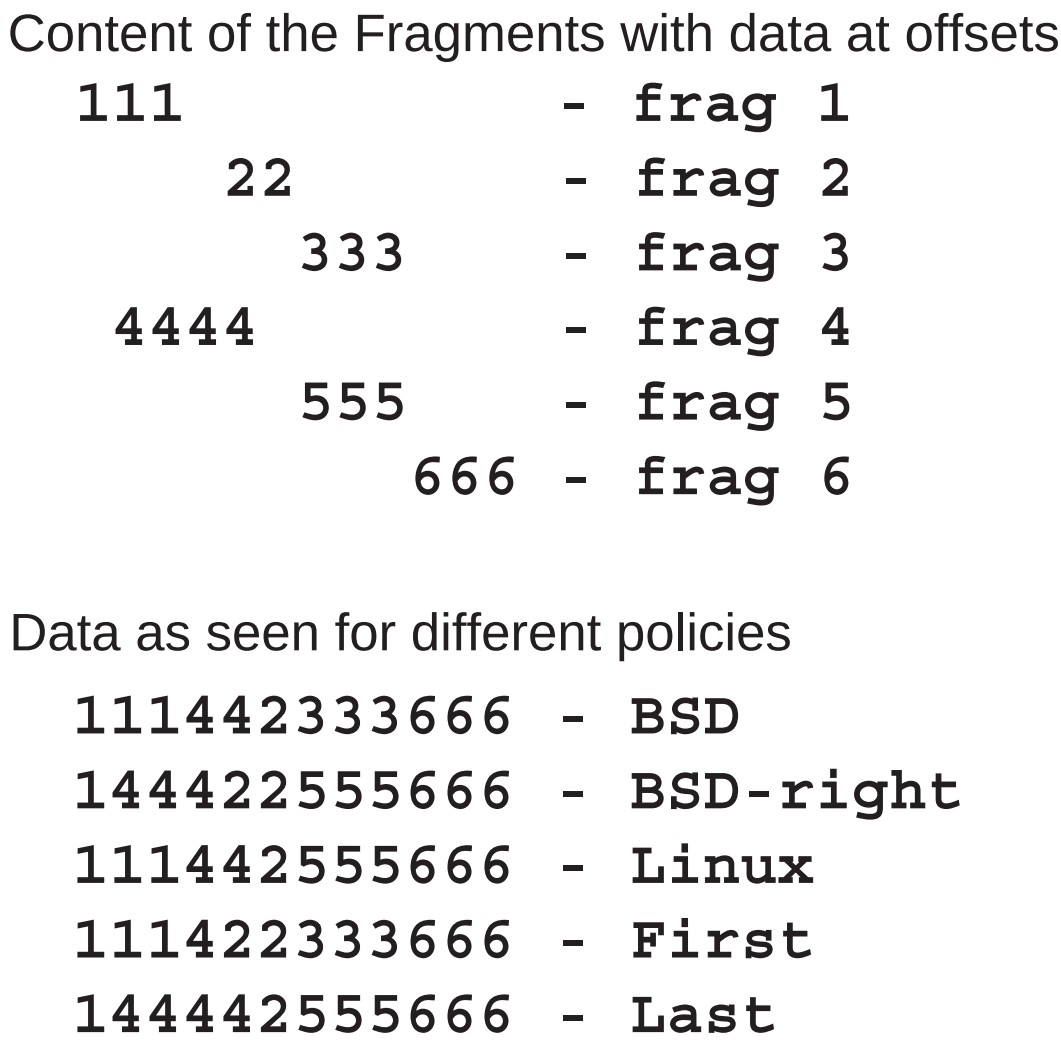


Figure 3.2: Data received for overlapped and incosistent fragment [14]

3.3 Denial of Service Attack on IDS/IPS

Denial of Service Attack can be launched on the IDS/IPS in many ways if the sessions are not maintained properly. Attack can be launched on the data structure used to maintain the TCP connections by opening too many connections. Also a

DOS attack can be launched on the buffer used to store the on-hold packets. A denial of service attack can also be launched by choosing the connection is such a way that all of them hash to the same value.

Chapter 4

TCP Reassembly

In this chapter we describe the Reassembly of TCP Streams from the individual packets to avoid attack by splitting of signature over multiple packets. In Section 4.1 we describe the data structure used to maintain the TCP stream data. In Section 4.2 Design of TCP Reassembly Module is discussed. Finally, incorporation of TCP Reaasembly Module in Sachet IPS is described in Section 4.3.

4.1 Data Structure

For maintaining the TCP connection data, a hash table is being used. Hash is computed on the quadruple:

- Source IP Address
- Source Port
- Destination IP Address
- Destination Port

Each entry in the hash table has a pointer to a linked list. All the entries that hash to the same value are in that linked list. To avoid data corruption in hash table we have a very basic locking mechanism. This is done using the bittestandset which is an atomic operation in SRAM. The most significant bit of the pointer in the hash table is used as lock bit for that entry. Whenever a thread operates on a stream it sets the lock bit and continues its work for that stream. Lock bit is reset once the work is finished on that entry. If the bit is already set, the thread leaves the control of the microengine voluntarily. When this thread regains control of the microengine it again checks for lock bit. This mechanism never leads to starving of a thread because all threads that are not waiting for any signal are scheduled in Round Robin manner by the hardware scheduler in the microengine.

As we have a 16 bit hash, the hash table have 64K entries. As each entry takes 4 bytes, the total size of hash table is 256KB. For each TCP connection we store data shown in the Table 4.1:

Field	Size(bytes)
Source IP	4
Destination IP	4
Source Port	2
Destination Port	2
Next TCP Sequence Number	4
Current FSM State	4
Time Stamp	8
TCP Connection State	2
Number of Onhold Packets	2
On-hold Packet Pointer	4
Next Connection Pointer	4

Table 4.1: Structure of TCP Connection Entry

We can perform signature detection only on the packets that are inorder. Packets coming out of order need to be kept in a buffer before they can be analysed by the signature detector. We refer to these packets as On-Hold Packets. The receiver module keeps all the incoming packets in the DRAM and passes a handle to the process module. Thus, TCP Reassembler stores the handle and other important information of all onhold packets for a connection in a linked list. A pointer to this linked list is stored in the TCP Connection entry of that connection. Information stored about onhold packets is shown in table 4.2

Field	Size(bytes)
Packet Handle	4
Buffer Length	4
TCP Sequence Number	4
Next Onhold Packet Pointer	4

Table 4.2: Structure of On-hold Packet Entry

Considering the amount of memory available, we have chosen to allow a maximum of 16K connection at a time. Each TCP stream entry takes 40bytes, so total memory required is 640KB. For storing onhold packet information we require 16 bytes per packet. We have chosen a maximum of 64K packets, ie average of four on-hold packets per connection. So total memory required in SRAM is 1MB. The actual packets are kept in DRAM. As maximum transmission unit of a packet is 1500 bytes. The maximum memory required in DRAM would be 93.75MB.

Table 4.3 shows the memory assignment to store all the data structures.

Data Structure Stored	Memory Type	Start Address	Size(bytes)
Aho-Corasick FSM	Local Memory	0	2560
Aho Corasick FSM	SRAM 0	100000	3M
Aho Corasick FSM	SRAM 1	40066000	3624K
Rule Mask Table	SRAM 0	66000	552K
Rule Constraint Table	SRAM 0	F0000	64K
Hash Table	SRAM 1	400C0000	256K
TCP Connection Entries	SRAM 1	40100000	640K
On-hold Packets Data	SRAM 1	40200000	1M

Table 4.3: Memory Assignment for Data Structures

4.2 TCP Reassembly Module

TCP Reassembly module takes a packet and validates its header. Header is checked for abnormal size packets, IP fragmentation, packets with IPV4 options, or packets of protocols other than IPV4. If the header validation fails then the packet is forwarded to port 2 of the network card. After successful header validation, it checks the transport layer protocol in the Protocol field of the IP Header, if the protocol is UDP or ICMP, the packet is simply forwarded for Signature Detection. If the protocol is TCP then we need to reassemble the stream.

TCP Reassembly module computes a 16-bit hash for the quadruple using the CRC unit present in the hardware and checks if there exists any entry for this connection. If the entry does not exist then a new entry needs to be created in the hash table. Before doing so we check if it is a SYN packet, if it is then an entry is created in the table else the packet is dropped. However, if the entry exists previously then it checks whether it is the next expected packet by looking at the TCP sequence number. If it is the next expected packet then sends it to Signature Detection module for scanning else stores its packet handle in a linked list for this stream. To avoid DOS attacks we have conservatively chosen not to allow more

than 8 on-hold packets for a single connection.

After the signature detection is done and if the packet is not clean then the packet is dropped and the entire stream data is deleted from the hash table and the buffer. If the packet is a FIN or a RST packet then the packet is forwarded and the stream entry is deleted from the TCP stream table. Otherwise, the TCP stream entry is updated and the packet is forwarded. Forwarding the packet as soon as it is scanned provides better throughput and also lesser usage of memory. There is no security risk in this even if the packet has a component of the signature because for an attack to be effective full signature needs to be present.

Pseudo Code for TCP Reconstruction Module is as follows. It makes a call to Signature Detection Module described in [5].

```
TCP Reconstruction
Start:
Read the packet header of new packet in microengine registers
Check for abnormal size packets, fragmentation, 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
  Calculate the hash key for the flow
  If SYN Packet
   If Entry exists in the hash table
   Drop Packet and goto Start
   Else
   Create a new entry in the hash table
   Set current FSM state = 0, i.e. start state
   Endif
  Else
   If Entry exist in the hash table
   Check the TCP if its the next expected packet. If yes continue
```

```
     else store the packet handle in the On-hold packets linked
     list and goto Start
Set current FSM state to stored current state of the stream
Else
Drop Packet
Endif
Endif

SigMatch:
Call Signature Matching
If there is a match clear the TCP stream entry from the hash table
  and drop all packets of this stream
Endif
If it is a FIN or RST Packet
Remove the TCP stream entry from hash table
Else
Update the TCP stream entry in the hash table
If next expected packet is in on-hold
Goto SigMatch
Endif
Endif
```

4.3 Incorporation in Sachet IPS

TCP Reassembly is implemented on the same microengine as the Signature Detection as both of them are tightly coupled. No modifications are made to the receive and transmit module.

Signature detector was enhanced to return the last state of the FSM after a packet is processed. This is then stored by the TCP Reassembler in the TCP

Connection Entry of that packet. It was also enhanced to allow start state to be different from state 0. For a TCP Connection state 0 is start state by defualt only for the first packet of that connection. For any subsidary packets the start state of FSM is the state that we stored in the TCP Connection Entry.

To avoid Denial of Service attack we have added garbage collection functionality. Whenever a thread sees that all the connections are exhausted it drops this connection and goes into garbage collection mode. It scans through all the TCP connection entries and drops those entries for which connection is not established properly. It even drops those connections for which no packet is received for a long period of time. We have chosen this period to be 1 hour.

Figure 4.1 shows the movement of packet inside the Sachet Intrusion Prevention System.

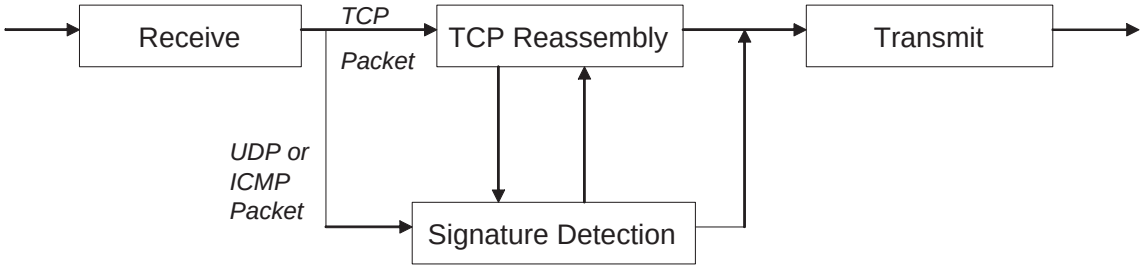


Figure 4.1: Packet Flow in Sachet IPS

Chapter 5

Web based Graphical User Interface

Sachet Console provides a Graphical User Interface to the system administrator for controlling the entire system. But it needs to reside on the same machine as the Sachet Server. Web based Graphical User Interface was added to Sachet so that the administrator can view the status of the Intrusion Detection System from any machine in the network rather than just the server. Apache web server is run on the same machine as the Sachet server. The user can connect to the web server from any machine in the network using a web browser. It needs to authenticate itself using a passphrase. After the user authenticates, a session is maintained for the user which times out if there is no activity for some time. The timeout period is specified in the configuration file on the server. The WebGUI connects to Sachet Server to obtain the requested data and displays it to the user. It has following screens:

- Login Screen - It asks for the login and password before the user can proceed to view information about the IDS.
- Agent - This screen shows the status of all the agents, including special agents like Learning Agent, Correlation Agent and Firewall Agent that are configured

in Sachet. It also allows to add and remove agents. Details about each agent can be viewed in a separate screen. It allows the user to view the status and alerts generated by that particular agent.

- Options - This screens allows the user to change database settings.
- Alert - This screen shows all the alerts generated by the agents.
- Signatures - List of signatures present in the database.

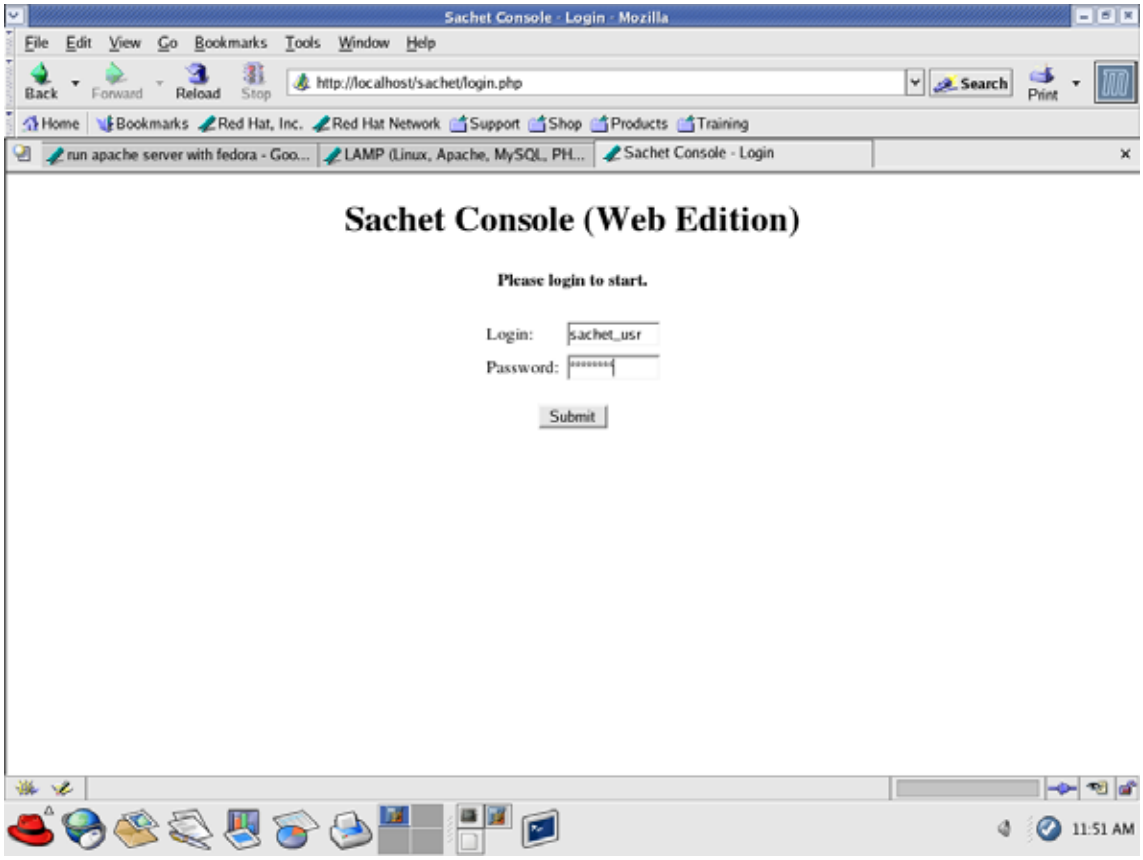


Figure 5.1: Main Screen to login into the system

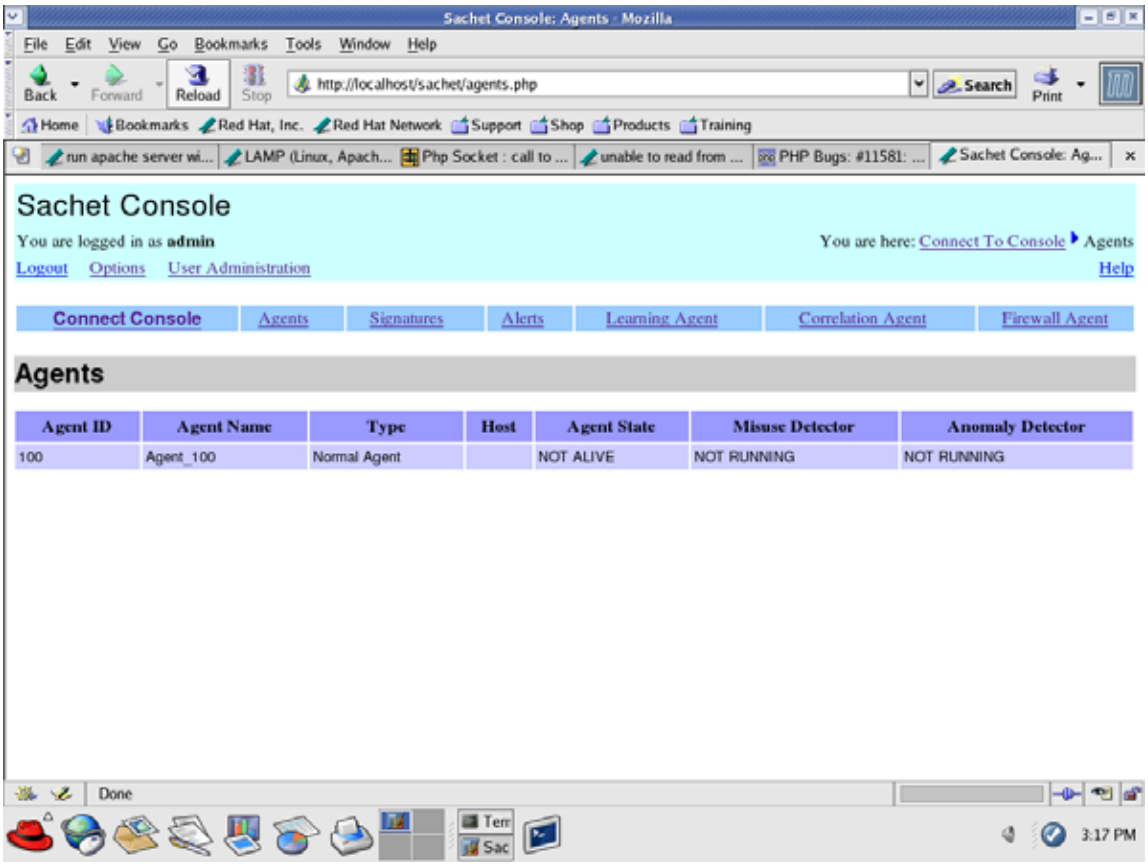
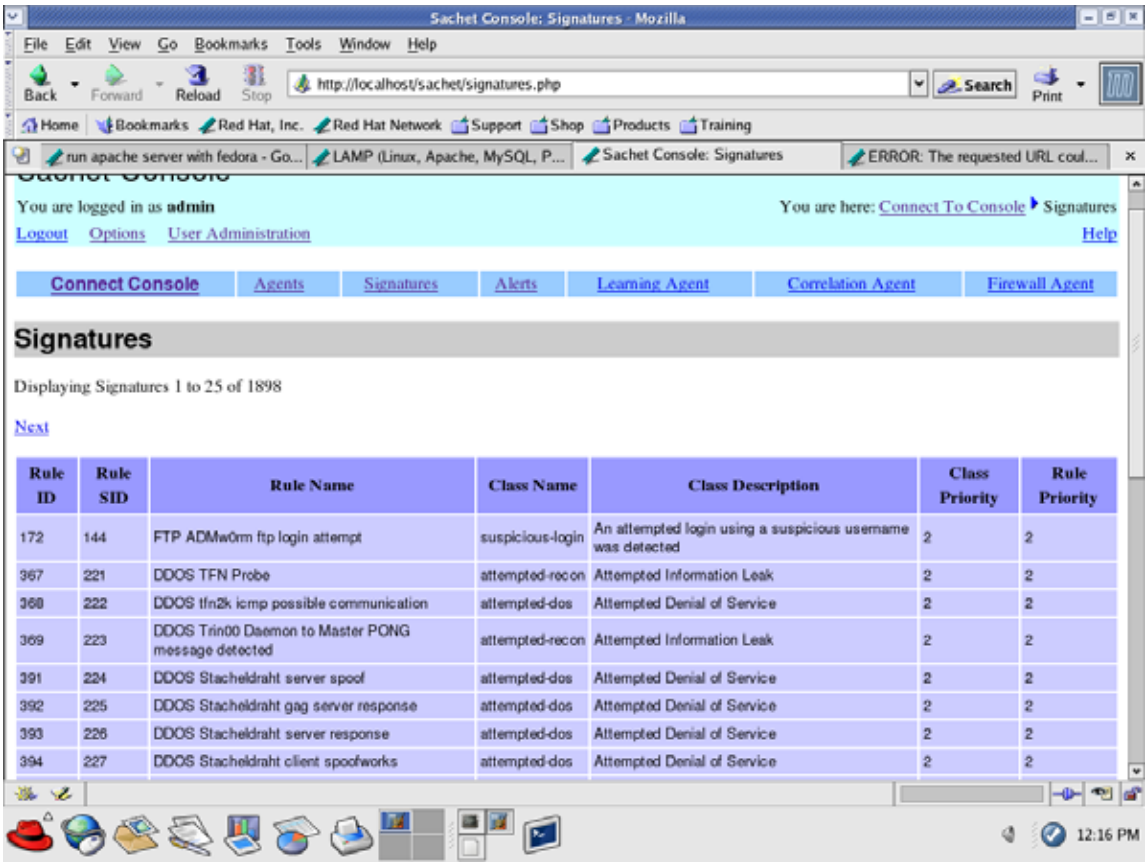


Figure 5.2: Screen to display the status of the agents



Rule ID	Rule SID	Rule Name	Class Name	Class Description	Class Priority	Rule Priority
172	144	FTP ADMw0rm ftp login attempt	suspicious-login	An attempted login using a suspicious username was detected	2	2
367	221	DDOS TFN Probe	attempted-recon	Attempted Information Leak	2	2
368	222	DDOS tfn2k icmp possible communication	attempted-dos	Attempted Denial of Service	2	2
369	223	DDOS Trin00 Daemon to Master PONG message detected	attempted-recon	Attempted Information Leak	2	2
391	224	DDOS Stacheldraht server spoof	attempted-dos	Attempted Denial of Service	2	2
392	225	DDOS Stacheldraht gag server response	attempted-dos	Attempted Denial of Service	2	2
393	226	DDOS Stacheldraht server response	attempted-dos	Attempted Denial of Service	2	2
394	227	DDOS Stacheldraht client spoofworks	attempted-dos	Attempted Denial of Service	2	2

Figure 5.3: Screen to display the list of signatures

Chapter 6

Conclusions and Future Work

A TCP Reconstruction module is developed for Sachet Intrusion Prevention System on an Intel IXP2400 network processor. This allows detection of signatures that are split into multiple packets of a TCP stream which may have gone undetected previously. This is an essential feature in an IDS/IPS. It is tested on the DARPA data set [1]. The system developed by us has been tested upto a data rate of 15 Mbps without dropping any packet. Before implementation of the TCP Reassembly module, the system could operate at data rates upto 24 Mbps. Therefore, better signature detection comes at the cost of reduction in maximum data rate that can be handled by the system. There is a maximum limit of 16K connections and 64K on-hold packets at a time, but this could be overcome by adding more memory to the network card.

In the current system, it is also assumed that the IP packets are not fragmented. A new layer needs to be added with the receiver module to defragment the fragmented IP packets before sending them for TCP Reassembly and Signature Detection. Currently all fragmented packets are dropped.

In this architecture, an attacker can also launch a Denial of Service attack on the system by opening many connections which hash to the same value. This is

because the number of possible connections that have the same hash value is high. To prevent this, a new layer needs to be added before TCP Reconstruction that filters out packets on the basis of IP addresses and TCP ports.

A Web based Graphical User Interface has been developed for Sachet IDS. It provides an additional functionality of allowing the network administrator to view the status of the system from any machine rather than just the server. The functional testing is performed by running an Apache server on the same machine as the Sachet Server. Status of the agents and alerts generated could be viewed from another machine on the network. Presently it has only one type of user. Adding multiple users with each having different level of access would enhance its utility.

Bibliography

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

[2] Nessus, open source network vulnerabiliyt scanner. *http://www.nessus.org/*.

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

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

[5] Asarawala, C. N. Intrusion Prevention and Automated Response in Sachet Intrusion Detection System. Master's thesis, Indian Institute of Technology, Kanpur, May 2006.

[6] Ben-Hur, A., Horn, D., Siegelmann, H., and Vapnik, V. Support vector clustering. *Journal of Machine Learning Research 2* (2001), 125–137.

[7] Bos, H., and Huang, K. Towards software-based signature detection for intrusion prevention on the network card. In *Proceedings of Eighth International Symposium on Recent Advances in Intrusion Detection (RAID2005)* (Seattle, WA, September 2005).

[8] Clark, C., Lee, W., Schimmel, D., Contis, D., Kone, M., and Thomas, A. A hardware platform for network intrusion detection and prevention. *Proceedings of The 3rd Workshop on Network Processors and Applications* (2004).

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

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

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

[12] 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.

[13] Ptacek, T. H., and Newsham, T. N. Insertion, evasion, and denial of service: Eluding network intrusion detection. Tech. rep., Secure Networks, Inc., 1998.

[14] Shankar, U., and Paxson, V. Active mapping: Resisting nids evasion without altering traffic. In *Proceedings of the 2003 IEEE Symposium on Security and Privacy* (Washington, DC, USA, 2003), IEEE Computer Society, p. 44.

[15] Vitter, J. S. Random sampling with a reservoir. *ACM Transactions on Mathematical Software 11*, 1 (Mar. 1985), 37–57.