Table of Contents

CHAPTER 1

INTRODUCTION

With the tremendous growth of network-based services and sensitive information on networks, network security is getting more importance than ever. Security is a big issue for all networks in today's enterprise environment. Hackers and intruders have made many successful attempts to bring down high-profile company networks and web services. Many methods have been developed to secure the network infrastructure and communication over the Internet, among them the use of firewalls, encryption, and virtual private networks. Intrusion detection is a relatively new addition to such techniques. Intrusion detection methods started appearing in the last few years. Using intrusion detection methods, we can collect and use information from known types of attacks and find out if someone is trying to attack your network or particular hosts. The information collected this way can be used to harden your network security, as well as for legal purposes. Both commercial and open source products are now available for this purpose. Many vulnerability assessment tools are also available in the market that can be used to assess different types of security holes present in your network. Although a wide range of security technologies such as information encryption, access control, and intrusion prevention can protect network-based systems, there are still many undetected intrusions. For example, firewalls cannot prevent internal attacks. Thus, Intrusion Detection Systems (IDSs) play a vital role in network security.

Network Intrusion Detection Systems (NIDSs) detect attacks by observing various network activities. The intrusion detection system, we made is NIDS. An IDS usually does not affect the normal network operations of the targets.

1.1 Purpose and Background

The number of information warfare attacks is increasing and becoming increasingly sophisticated. Annual reports indicate a significant increase in the number of computer security incidents each year. Not only are these attacks becoming more numerous, they are also becoming more sophisticated. The reports show the growing use of "widespread attacks using scripted tools to control a collection of information gathering and exploitation tools". Likewise it reports the growing use of automated scripts that launch

and control tens of thousands of attacks against one or more targets. Each attacked computer has limited information on who is initiating the attack and from where. The threat of a sophisticated computer attacks is growing. Unfortunately, intrusion detection and response systems have not kept up with the increasing threat.

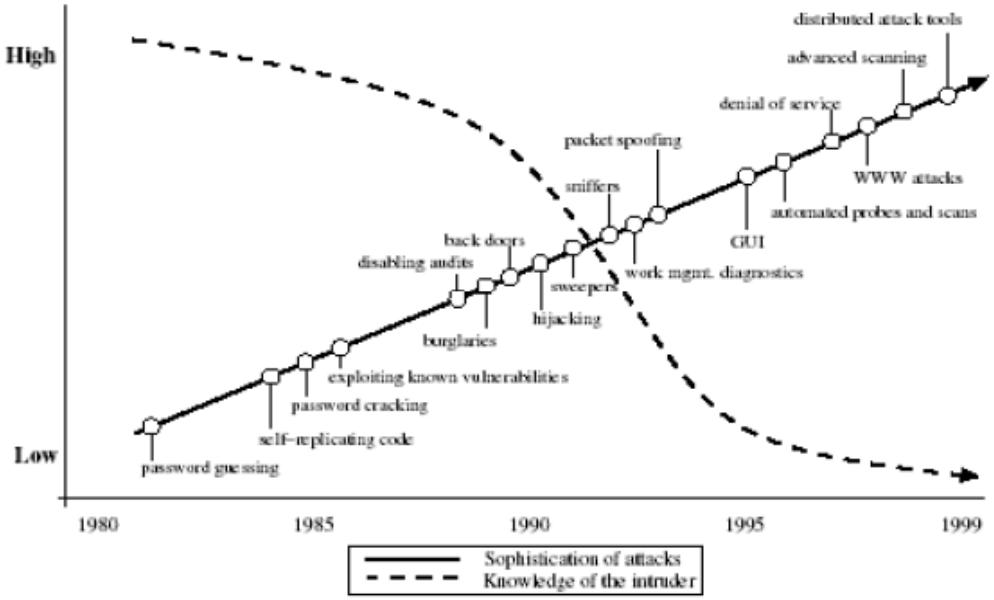

Figure 1: The increase in sophistication of attack and decrease in expertise of users

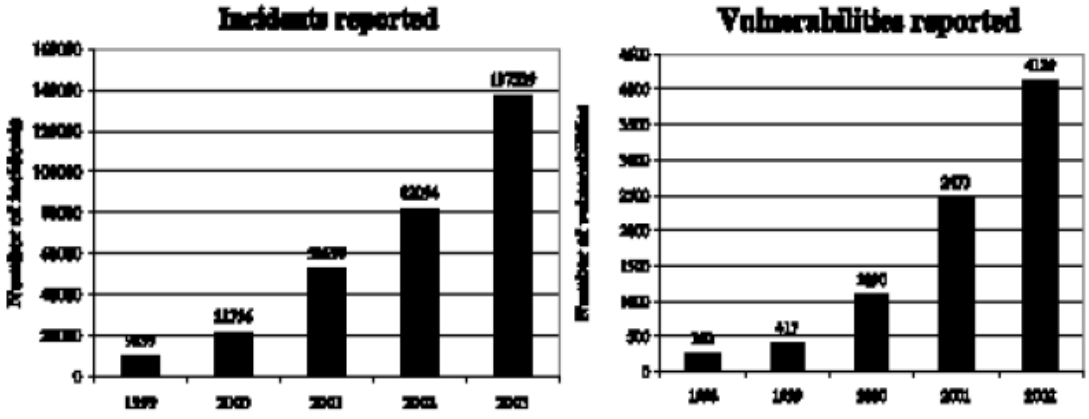

Figure 2: The number of incidents and vulnerabilities reported per year(CERT)

This delay response, ranging from minutes to months, provides a window of opportunity for attackers to exploit. F.B. Cohen explored the effect of reaction time on the success rate of attacks using simulations. The results indicate that if skilled attackers are given ten hours after they are detected before a response, they will be successful 80% of the time. If they are given twenty hours, they will succeed 95% of the time. At thirty hours, the attacker almost never fails. The simulation results were also correlated against the skill of the defending system administrator. The results indicate that if a skilled attacker is given more than thirty hours, the skill of the system administrator becomes irrelevant - the attacker will succeed. On the other hand, if the response is instantaneous, the probability of a successful attack against a skilled system administrator is almost zero. Response is a fundamental factor in whether or not an attack is successful. Attackers would simply adapt their approach so as to mediate the defense. An adaptive automated intrusion response system provides the best possible defense by notifying system administrator against the attack or takes the action associated with the attack.

1.2 Block Diagram of the project

Our project consists of different components as shown in the below diagram (figure 3). From the network packet is sniffed. The sniffed packet is taken by sensor agent and compared with the existing rules, with the help of PackageHandler and RuleParser. If all the fields of the rules and the packet are matched then intrusion is detected which is logged in file and the description of the intrusion in the rule field is send to response agent. The response agents do appropriate action with the help of response table. These activities seen are handled by the MonitorAgent, which is displayed with the help of Graphical User Interface (GUI) to the Network Administrator.

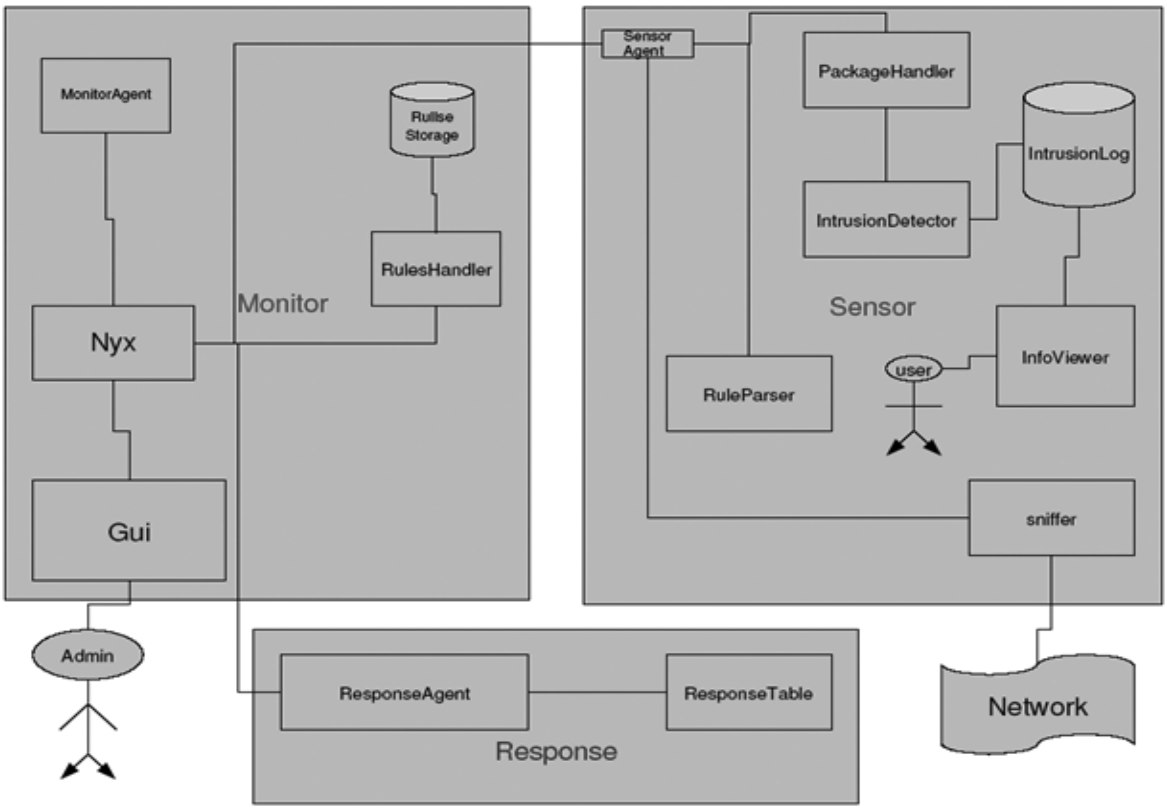

Figure 3: Block Diagram

1.3 Data Flow Diagram (DFD)

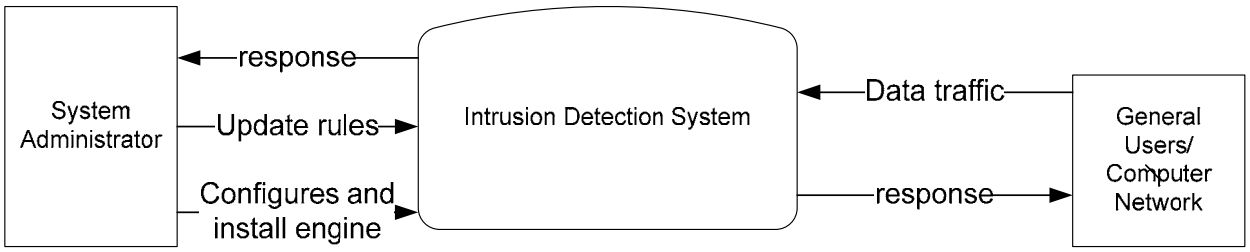

Figure 4: Context diagram of the project (level 0)

The context diagram shows how the Intrusion Detection System interacts with the external users. System administrator configures and installs the engine in the network where IDS is to be kept. He updates rules. IDS will respond towards the system administrator if any intrusion is detected. The next users are general users who may not be aware about the IDS. IDS will get traffic from computer networks (general users' workplace). After sniffing packets it will processes the packet and reacts with appropriate action to the general users as to the system administrator.

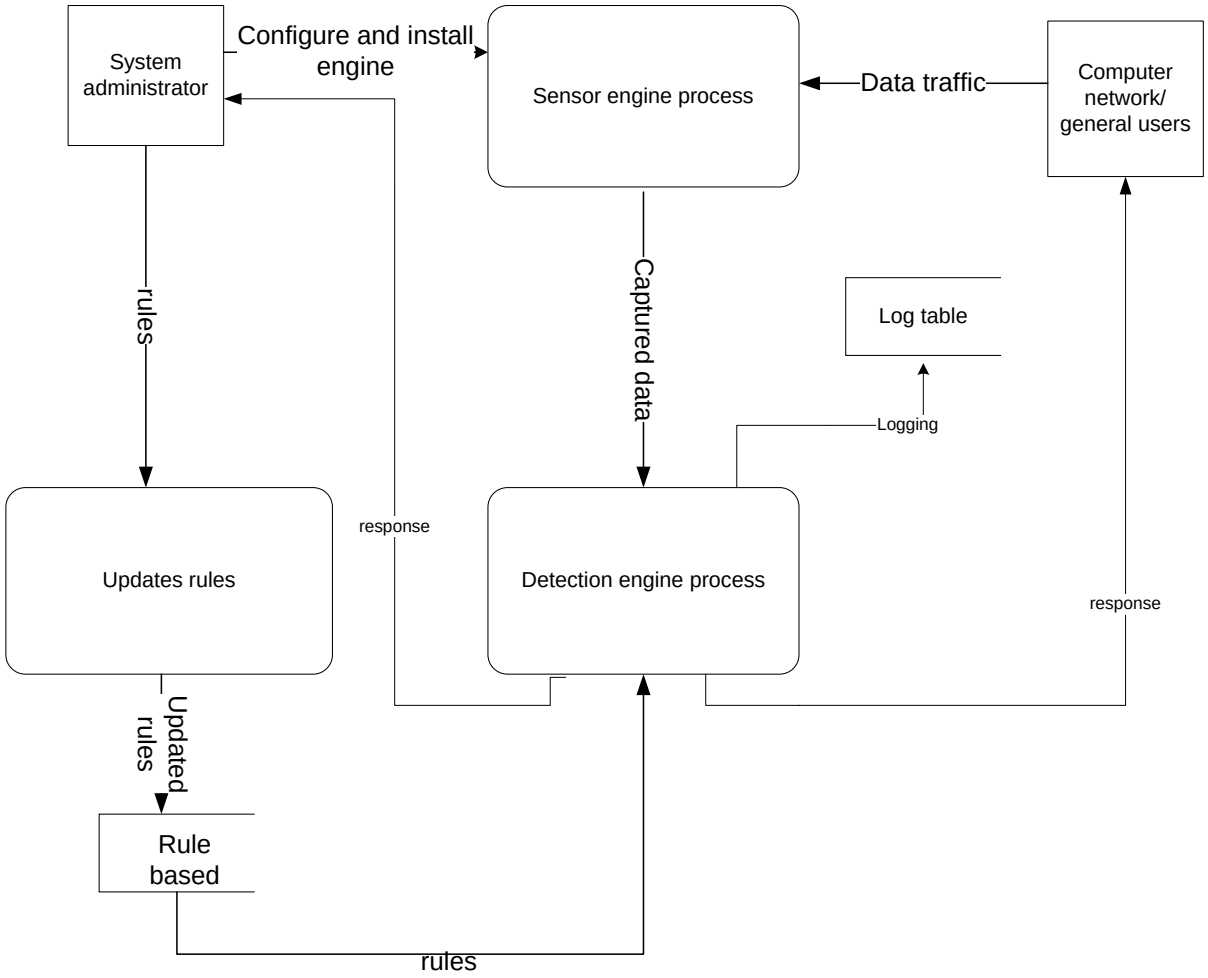

Figure 5: Level 1 DFD of the system

Level 1 DFD shows the detail operation of the previous context diagram. System administrator configures and installs sensor engines process. He updates rules in the rule based. Sensor engine process sniffs data packets from the computer networks. The captured data is sent to the detection engine process. This detection engine will process the packet and compares with the rules and find if the activity is intruder. The intrusion activity is logged in the log table. This log table is used for generating reports for the IDS. The detection engine process responses to the system administrator and general users about the intrusion by alerting them and taking proper actions.

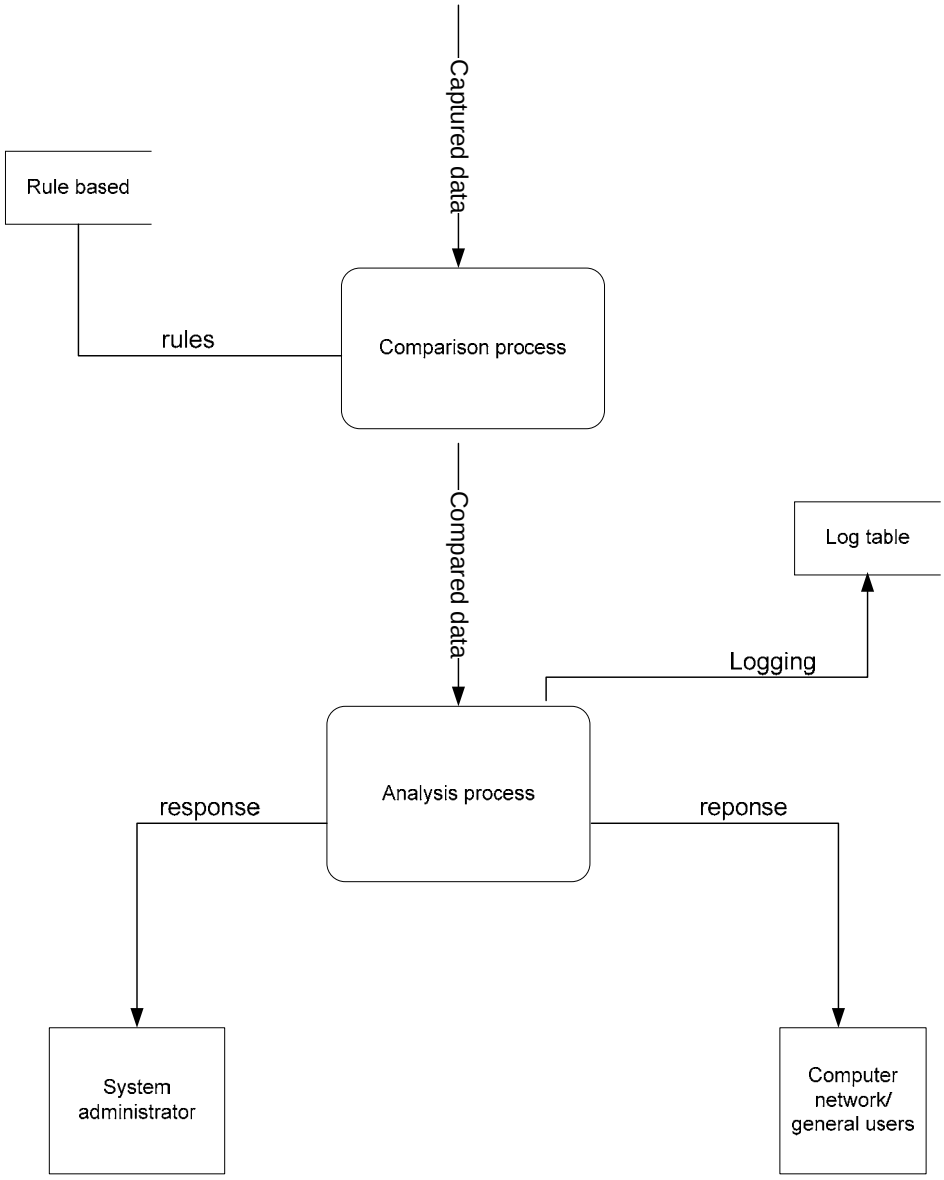

Figure 6: Level 2 DFD of the detection engine process.

The captured data from the sensor engine process is processed by the comparison process. This process compares with the rules from the rules based storage. The compared data is analyzed by the analysis process. Analysis process will response to the system administrator and to the general users. The activity of the intrusion is logged in the log table. In this way detection engine process is broken into comparison and analysis process.

1.4 Objectives of the Project

Every research works and project works are performed to achieve a goal. So as to achieve the goal, the objective of the project needs to be clearly defined. Hence the following points explain the objectives of this project.

a) To develop intrusion detection system using agents.

b) To protect secure information of an organization from outside and inside intruders

c) To provide a secure environment for work.

d) To decrease the bottleneck in the main server by distributing the sensors in the particular hosts.

e) The log of the intrusion detection system can be used for forensic purpose to find out the culprit.

1.5 Scope of the Project

The project finds its scope in any organization that has valuable resources kept in computer network. These days information is the most valuable and powerful tool. This information is also becoming vulnerable to the hackers, intruder, etc who can make damage to the network. This project deals with one of the main features of computer network security. These days huge amount of money is expended in computer security system. We can talk about the example of China which is expensing a huge billion of money for its computer security.

Intrusion detection system is needed to the following bodies:

a) **Government authorities**: Government authorities will have lots of information about intangible and intangible properties. They keep data about the people on the basis of different aspects of living to make proper strategic planning. The storage and terminals of the government's IT works should be secured and out of reach of

intruders. So, Intrusion Detection System implemented in such computer networks will be securing the valuable assets.

b) **Banks and financial institutions**: The property of a people is said if it is kept in banks. Only digital record is given to the person such as account number, ATM cards etc to access his/her property. If the digital format is leaked or hacked then the existence of Banks cannot be imagined. We are hearing in news about huge amount loss in the hacking of Recharge number codes of mobile company, ATM cards in our country Nepal too. Intrusion Detection System will certainly minimize the unauthorized access and take immediate response to stop such illegal works.

c) **Multinational companies**: Multinational companies have management through Software. The merging of multinational companies is due to their all transactions and reports in digital format. This information are passed through different networks and done through secure layers or different security techniques. But still they are prone to intruder attacks. So intrusion detection system is very essential to the networks of such multinational companies' computer networks.

d) **Police Department**, **Army Department**, **Intellectual Agencies** have valuable information in computers. They have very valuable and sensitive information. This information should be kept secretly. So, the computer networks and terminals should have intrusion detection system for the safety of the information against the probable intruders.

Not only this much, intrusion detection system is needed to all universities, educational sectors too. Most of the examinations are done in digitally transferable medium. The election voting is also done and counted through computer medium. All these information are to be secured and must be alerted with possible intruders.

Chapter 2

LITERATURE REVIEW

2.1 Internet Protocol (IP) Concept:

The Internet Protocol (IP) is a large and potentially intimidating topic that requires a gentle introduction for uninitiated beginners so as not to overwhelm them with foreign acronyms, details, and concepts. Therefore, the purpose of this first chapter is to expose newcomers to terms, concepts, and the ever-present acronyms of IP. The suite of protocols covered here is more commonly known as Transmission Control Protocol/Internet Protocol (TCP/IP). These protocols are required to communicate between hosts on the Internet—the worldwide infrastructure of networked hosts. Indeed, communication protocols other than TCP/IP exist (for instance, AppleTalk for Apple computers). These protocols are typically found on intranets, where associated hosts talk on a private network. Most Internet communications require TCP/IP, which is the standard for global communications between hosts and networks. This is an around-the-world introduction to TCP/IP.

The TCP/IP Internet model. This section examines the foundations of communications over the Internet, specifically communications made possible by using a common model known as the TCP/IP Internet model.

Packaging of data on the Internet. This section reviews the encapsulation of data to be sent through different legs of a journey to its destination.

Physical and logical addresses. This section highlights the different ways to identify a computer or host on the Internet.

TCP/IP services and ports. This section explores how hosts communicate with each other for different purposes and through different applications.

2.1.1 The TCP/IP Internet Model

Computer users often want to communicate with another computer on the Internet for some purpose or another (to view a web page on a remote web server, for instance). A response from a web server can seem almost instantaneous, but a lot of processes and infrastructures actually support this seemingly trivial act behind the scenes.

Layers

Figure 7 shows a logical roadmap of some of the processes involved in host-to-host communications. You begin the process of downloading a web page in the box labeled

"Web browser." Before your request to see a web page can get to the web server, your computer must package the request and send it through various processes and layers. Each layer represents a logical leg in the journey from the sending computer to the receiving computer. After the sending computer packages the data through the different layers, it is delivered to the receiving computer over the Internet. The receiving computer unwraps the data layer by layer. An individual layer gets the data intended for it and passes the remainder of the message to upper Layers.

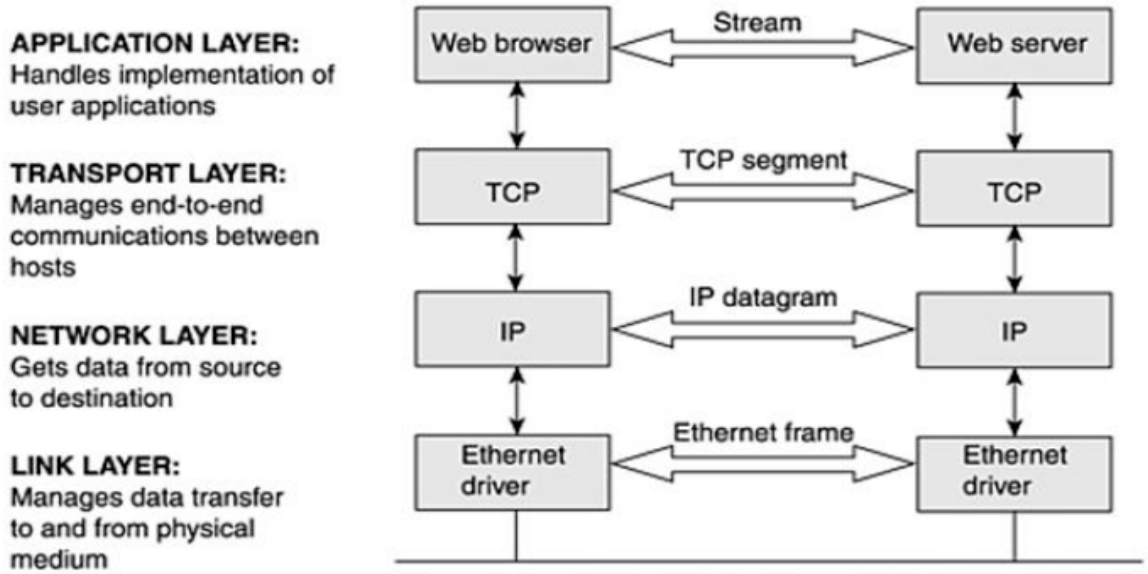

Figure 7: The TCP/IP Internet model.

Data Flow

In theory, the data flow activity is this: The request for a web page "descends" the sender's layers, often referred to as the TCP/IP stack. It gets directed to the destination computer and "ascends" its TCP/IP stack. The vertical arrows between layers represent the up and down flow on the same computer. The horizontal arrows between computers signify that each layer talks to its "peer" layer on the communicating host. The two computers do not directly interact with each other, per se. When the request descends the sending computer's TCP/IP stack, it is packaged in such a manner that each layer has a message for its counterpart layer, and so they appear to be talking directly. This concept is quite important and crucial to understanding this chapter and the TCP/IP model, in

general. Therefore, it is important to reiterate the poignant points and elaborate on terminology. The term TCP/IP stack is used to denote the layered structure of processing a TCP/IP request or response. A process known as encapsulation does the implementation of the layering. This means that data on the sender's host gets wrapped with identifying information to assist the receiving host in parsing the received message layer by layer. Each layer on the sending host adds its own header, and the receiving host reverses the process by examining the message, stripping it of its header, and directing it to the appropriate layer. This process is repeated for the higher layers until the data reaches the uppermost layer, which finally processes the web page request. When the response is sent back, the entire process is repeated; now the web server host packages the data to be sent, it is delivered and received, and the web browser host strips the received message to pass to the application layer supporting the web browser.

2.1.2 Packaging (Beyond Paper or Plastic)

At a very granular level, data exchanged between hosts must be bundled in some kind of standard format. A host is a generic term that can reference a workstation on our desk, a router, or a web server to name just a few examples. The important distinction is that these computers are connected to a network capable of transporting data to and from the computer. In the generic sense, the packaging of associated data is called a packet. The problem in terminology arises because this data package is labeled differently at various layers of communication between the source application and the destination application located on different hosts. This section discusses some of the key concepts related to data packaging, including bits, bytes, packets, data encapsulation, and interpretation of the layers.

Bits, Bytes, and Packets

The atom of computing is a bit, a single storage location that has a value of either 0 or 1 (also known as binary). Although succinct and compact, we cannot actually store or convey a lot of information with a single bit, so bits are grouped into clumps of eight. A unit of eight bits is a byte. Eight times a very small amount of information is still pretty small, but an octet can contain an American Standard Code for Information Interchange

(ASCII) character, such as the letter a or a comma (,). It can also hold a large integer number, as high as 255 (28-1).

Interpretation of the Layers

With all the layering going on, the bottom line is that we have a bunch of adjacent 0s and 1s. The term *protocol* is meant to denote a set of agreed upon rules or formats. Each protocol (such as IP, TCP, UDP, and ICMP) has its own layouts and formats.

Figure 8 shows an example of the organization of the IP header. We can see that a certain number of bits are allocated for each field in the header. A Protocol field identifies the embedded protocol. Each row that we see in the IP header is 32 bits (0 through 31, inclusive), which means four (8-bit) bytes. To complicate matters a little, counting starts with 0 when talking about bit and byte locations. The first row represents bytes 0 through 3; the second row represents bytes 4 through 7; and the third row represents bytes 8 through 11. Notice that the circled Protocol field is in the third row. The preceding time-to-live (TTL) field is 1 byte long, which makes it the 8th byte; and the Protocol field, which is also 1 byte long, represents the 9th byte. This means that the 9th byte (actually, it's the 10th byte, but remember counting starts at 0) is examined to find the embedded protocol. The point is that most packets at their respective levels are positional; fields can be discovered by going to known displacements in the packet.

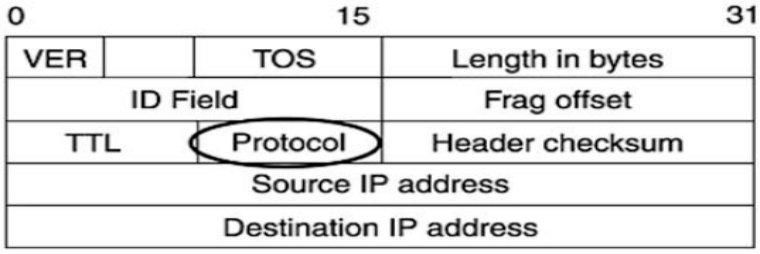

Figure 8: Positional layouts of TCP

Now that we have counted our way to the Protocol field, what is it and what does it do? The value in this field tells us what protocol is found in the embedded data. Suppose that the value we find in this byte is 17. We might find the protocol value expressed in hexadecimal. A hexadecimal 11 is the same as a decimal 17. This means that a UDP

packet is embedded after the IP header. A value of 6 means that the embedded packet is TCP, and a value of 1 means that it is Internet Control Message Protocol (ICMP).

2.1.3 Addresses

Most likely, we have heard the term IP address. But, what does it really represent and what does it really do? And, exactly how do hosts address each other? These are some of the topics covered in this section.

2.1.3.1 Physical Addresses, Media Access Controller Addresses

When we scour the headers of IP packets looking for physical layer MAC addresses until we turn blue, and we will not find them. MAC addresses do not mean anything to IP, which uses logical addresses; they are not part of the protocol. For all intents and purposes, they may as well not exist. By the same token, physical MAC addresses are how the Ethernet card interfaces with the network. The Ethernet card does not know a single thing about IP, IP headers, or logical IP addresses. Clearly, if things are going to work, an operation process is required that facilitates the correspondence between logical IP and physical MAC addresses. IP address space is a precious commodity. When we connect to the network, many of us are loaned an address for that session, or possibly longer by an Internet service provider (ISP) or network service provider via applications, such as Dynamic Host Configuration Protocol (DHCP).

2.1.3.2 Logical Addresses, IP Addresses

An IP address has 32 allocated bits to identify a host. This 32-bit number is expressed as four decimal numbers separated by periods (for example, 192.168.5.5). These are not just random or sequential assignments. The initial portion of the IP number tells something about the size of the network on which the host resides. The remainder of the IP number distinguishes hosts on that network. Addresses are categorized by class; classes tell how many hosts are in a given network or how many bits in the IP address are assigned for the unique hosts in a network. A grouping known as Class A addresses assigns the initial 8 bits for a network portion of the address, for example, and the final 24 bits for the host portion of the address. Because 24 bits have been allocated for the hosts, more than 16 million (224-1) hosts can possibly be in the network. An example of a Class A network is

the 18.0.0.0 through 18.255.255.255, IP space assigned to Massachusetts Institute of Technology.

Table 1: Bits for IP Address Space

Table 1.1. 32 Bits for IP Address Space			
Class	**Network Bits**	**Host Bits**	**Number of Hosts**
A	8	24	16 million+
B	16	16	65,000+
C	24	8	255

The IP address classes range from Class A addresses to Class E. Classes A, B, and C are unicast addresses; when we send a packet to them, presumably we are addressing a single machine. Class D is known as a multicast address used to communicate with a designated set of hosts. Class E is reserved for experimental use. Table 2 shows the address range associated with each class.

Table 2: Address Classes and IP Ranges

Class	Beginning IP	Ending IP
A	0.0.0.0	127.255.255.255
B	128.0.0.0	191.255.255.255
C	192.0.0.0	223.255.255.255
D	224.0.0.0	239.255.255.255
E	240.0.0.0	247.255.255.255

House Rules of CIDR

Classless inter-domain routing (CIDR) refers to addresses. For the longest time, addresses were part of a particular class and that meant our network was allocated either16 million+, 65,000+, or 255 hosts. The most common situation was networks that required between 255 and 65,000 hosts. Because many of these sites were allocated Class B networks, many IP numbers went unassigned. Given that IP numbers are finite commodities, a remedy was needed to allocate networks without class constraints. CIDR assigns networks, not on 8-bit boundaries, but on single-bit boundaries. This allows a site to receive the appropriate number of IP numbers, and thus reduces waste. CIDR uses a unique notation to designate the range of hosts assigned to a site. If we want to specify the 192.168 address range in CIDR, it would look like 192.168/16. The first part of the notation is the decimal representation of the bit pattern allocated to the network. It is

followed by a slash and then the number of bits that represent the network portion of the address. This example is the same as a Class B network, but it can be modified easily enough to represent smaller networks.

Subnet Masks

Another concept we need to be aware of is something known as the subnet mask. This mask informs a given computer system how many bits in its IP address have been relegated to the network and how many to the host. Each bit that is a network bit is "masked" with a 1. A Class A address, for instance, has 8 network bits and 24 host bits. In binary, the 8 consecutive bits (all with a value of 1) translate to a decimal 255. The subnet mask is then designated as 255.0.0.0. Other classes have other subnet masks. A Class B network has a standard subnet mask of 255.255.0.0, and a Class C network has a standard subnet mask of 255.255.255.0. Why is this needed if we can tell what class and how many bits have been reserved for the network by examining the IP address? Some network administrators subdivide their networks.

For instance, a Class C network could be divided into four individual subnets by assigning an appropriate subnet mask.

2.1.4 Service Ports

TCP and UDP have 16-bit port number fields in their respective header fields. This means they can have as many as 65,536 different ports, or services, and they are numbered from 0 to 65,535. One very important point to register in our long-term memory is that even though a service is usually located at its assigned port number, nothing guarantees this as true. Telnet, for instance, is almost universally found on TCP port 23. There is nothing stopping our nonconformist side from offering it at port 31337. And, what better way for a hacker who has broken into a computer to hide his tracks than by offering a service at an unexpected port? If a hacker were to run telnet at some high-numbered port rather than port 23, it would make his unauthorized connection more difficult to find and identify. Any service can be run at any port. On the other hand, if we want to network with other hosts, it is best to follow the standards. For UNIX hosts, the /etc/services file can be an excellent resource to match TCP or UDP port numbers with the expected, or well-known, services likely to be offered at that port number.

We see some very common port numbers and service examples from the /etc/services file. An excerpt here shows us the format of the file and the associated services. We see that a service known as domain (Domain Name Service, or DNS) can be offered on both TCP and UDP. This is unusual, but not abnormal; most services are offered on either TCP or UDP, but there are some exceptions (such as DNS).

ftp 21/tcp

telnet 23/tcp

smtp 25/tcp

domain 53/udp

domain 53/tcp

Figure 9 shows how the service is specified in the packet. In this case, a UDP header has a 16-bit field known as the destination port. This is where the desired service or port is found. In this example, the value in the UDP header's port number field would be 53, signifying that this datagram is destined for the Domain Name Service.

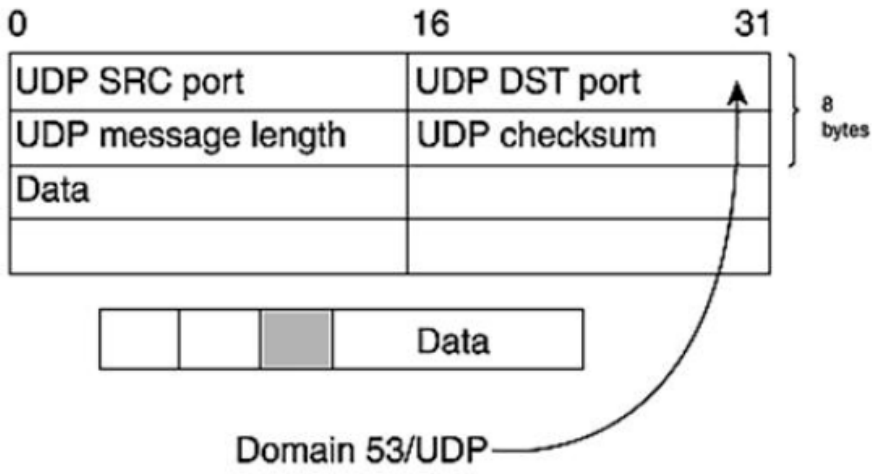

Figure 9: Not just any port.

At one time in history, special significance was attached to ports below 1024. Those lower-numbered ports were the so-called trusted ports (chuckle) because only root could use them. The term *trusted port* originated because ports below 1024 were allocated to system processes. Therefore, if a foreign host saw an incoming connection with a source port less than1024, it was assumed to be trusted because it ostensibly came from a system process. This made much more sense when the Internet was a safer place. This is much

less true today, but the ports above 1024 have special significance. These are often called the ephemeral ports, which mean that they could be used by most any service for most any reason.

2.2 Introduction to TCP

TCP is a reliable connection-oriented protocol used with well-known applications such as telnet or smtp. An application such as telnet cannot tolerate the uncertainty of the Internet Protocol that can lose datagrams or deliver them in a different order from which they were sent. TCP is the protocol that orchestrates and ensures reliability. It does so using the following mechanisms:

- **Exclusive TCP connection.** When a TCP session is established, the connection is exclusive and unique between the two hosts. This kind of connection is called a unicast connection. The negotiation of the unique session allows both sides to track the traffic exchanged between the two hosts.
- **TCP sequence numbers.** These provide a sense of chronology to the TCP data sent and received. A telnet command or exchange might take several packets known as TCP segments to transmit all the data. Data is assigned a TCP sequence number to uniquely identify the data in each segment being sent. Because the data might arrive in a different order from which it was sent, TCP sequence numbers are also used to reassemble the data in the correct order.
- **Acknowledgements.** Acknowledgements are used to inform the sender that data has been received. Acknowledgements are made to sequence numbers to identify the exact data received. If the sender does not receive an acknowledgement for specific data in a given time, it assumes that the data has been lost. The sender will retransmit what it believes was lost.

2.2.1 Server and Client Ports

In the past, more so than today, well-known server ports generally fell in the range of 1–1023. Historically under UNIX, only processes running with root privilege could open a port below 1024. These ports should remain constant on the host for which they are offered. In other words, if we find telnet at port 23 on a particular host one day, we

should find it there the next day. We will find many of the older well-established services in this range of 1–1023 (such as telnet on port 23 and smtp on port 25). Today, some of the newer services, such as AOL Instant Messenger, usually associated with TCP port 5190, don't tend to conform to this original convention. This is partially because there are more services than numbers in this range today.

Client ports, often known as *ephemeral ports*, are selected only for a particular connection and are reused after the connection is freed. These are generally numbered greater than 1023. When a client initiates a connection to a server, an unused ephemeral port is selected. For most services, the client and server continue to exchange data on these two ports for the entirety of the session. This connection is known as a *socket pair* and it will be unique. There will be only one connection on the Internet that has this combination of source IP and source port connected to this destination IP and destination port. Someone from the same source IP might even be connected to the same destination IP and port. This user will be given a different ephemeral port, however, thus distinguishing it from the other connection to the same server and destination port. Two users on the same host might connect to the same web server. Although this is the same source IP, destination IP, and port (80), the web server can maintain who gets what by the ephemeral source ports involved. Examine the three-way handshake exchange again, but this time in the context of client and server ports:

tclient.net.39904 > telnet.com.23: S 733381829:733381829(0) win 8760 <mss 1460> (DF)

telnet.com.23 > tclient.net.39904: S 1192930639:1192930639(0) ack 733381830 win 1024 <mss 1460> (DF)

tclient.net.39904 > telnet.com.23: . ack 1 win 8760 (DF)

We see that tclient.net has selected ephemeral port 39904 on which to communicate and to connect to well-known port 23 of telnet.com. Any further exchanges after the three-way handshake are done using these two negotiated ports. After the connection is closed and some time has passed, tclient.net releases port 39904 for use by another connection. Port 23 of telnet.com remains bound to the telnet service for additional telnet requests.

2.3 ICMP

I*nternet Control Message Protocol* (ICMP) was conceived as an innocuous method of reporting error conditions and issuing and responding to simple requests. Perhaps because of its seemingly benign origins, some of the current mutations of ICMP for less-than-upstanding purposes seem all the more outrageous. In its pure state, ICMP is supposed to be a relatively simple and chaste protocol, but it has been altered to act as a conduit for evil purposes. Therefore, it is important to understand how this protocol is used both for its intended purposes and for malicious purposes. We must know about both the expected and unexpected uses of ICMP that we might see in our own network. Then we must put this ICMP theory into action by analyzing some unusual detected ICMP activity. Finally, the discussion focuses on protecting our network by blocking inbound ICMP activity and the accompanying repercussions of doing so.

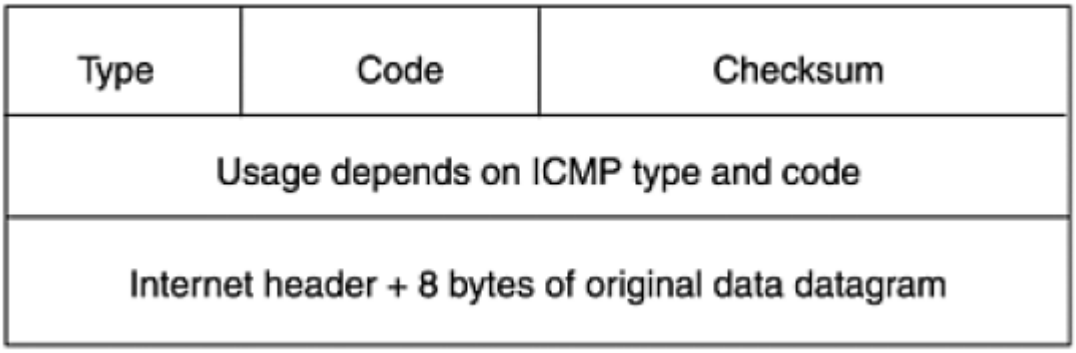

Figure 10: ICMP error message format

Normal ICMP activities are as follow:

- Host unreachable

  router > sending.host: icmp: host target.host unreachable

- Port Unreachable

  target.host > sending.host: icmp: target.host udp port ntp unreachable (DF)

- Admin Prohibited

  router > sending.host: icmp: host target.host unreachable - admin prohibited

- Need to Frag

  router > sending.host.net: icmp: target.host unreachable - need to frag (mtu1500)

- Time Exceeded In-Transit

  routerx > sending host: icmp: time exceeded in-transit

There are many malicious ICMP detected. One of them is explained below:

- Smurf Attack

The infamous Smurf attack, shown in Figure 11, preys on ICMP's capability to send traffic to the broadcast address. Many hosts can listen and respond to a single ICMP echo request sent to a broadcast address. This capability is used to execute a denial-of-service attack against a hapless target host or network.

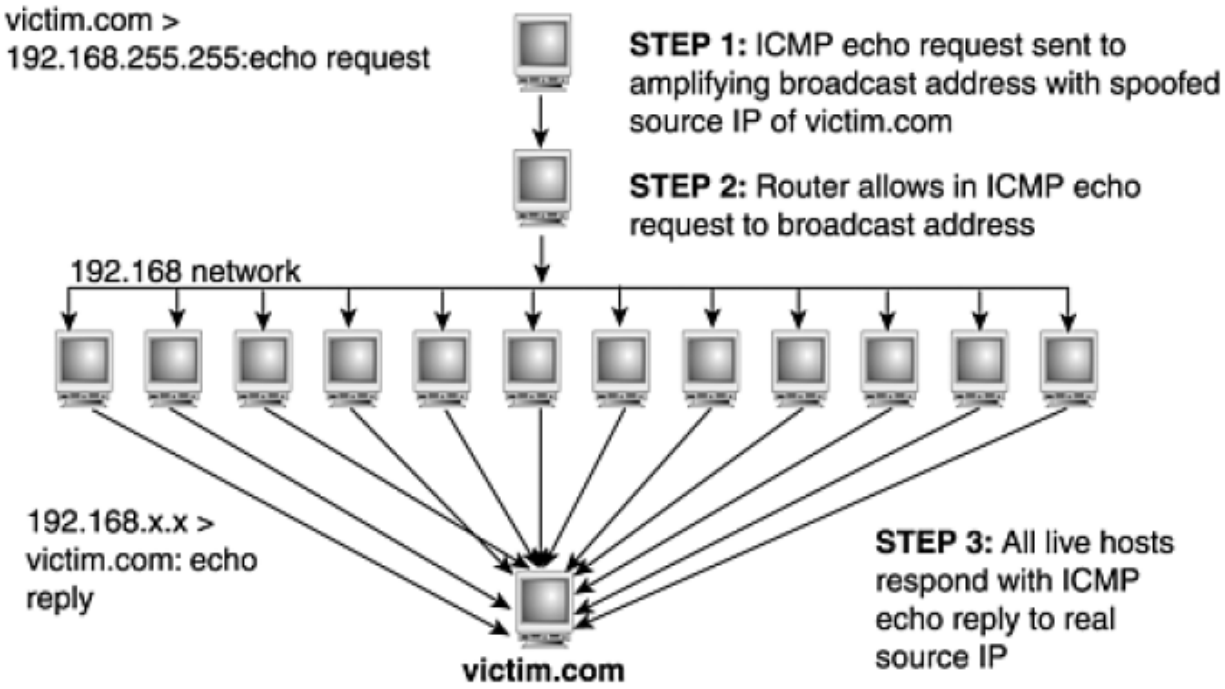

Figure 11: Anatomy of a Smurf attack.

First, a malicious host must craft an ICMP echo request with a spoofed source IP to a broadcast address of an intermediate network. The spoofed source IP chosen is that of the victim target host/network. Next, the intermediate site must allow broadcast activity into

the network. If it does, the ICMP echo request is sent to all hosts on the given subnet to which the broadcast was sent. Finally, all the live hosts in the intermediate network that respond send an ICMP echo reply to what they believe to be the sender, or the victim host. The victim host or network on which it resides can become choked with all the activity and can suffer a degradation or denial-of-service attack if the following conditions exist:

- The malicious user sends many ICMP requests to the broadcast address.
- The intermediate site allows inbound broadcast traffic.
- The intermediate site is large and has many responding hosts. On the other hand, many smaller intermediate sites might be used to achieve the same result.
- The target site has a slow Internet connection. To be more precise, the Internet onnection must be susceptible of being overwhelmed by too many packets for the supported bandwidth. Although it is possible to inundate and clog *any* Internet connection given enough traffic, slower connections are more vulnerable.

2.4 UDP

User Datagram Protocol (UDP) is one of the core protocols of the Internet protocol suite. Using UDP, programs on networked computers can send short messages sometimes known as datagrams (using Datagram Sockets) to one another. UDP is sometimes called the Universal Datagram Protocol. It was designed by David P. Reed in 1980.

UDP does not guarantee reliability or ordering in the way that TCP does. Datagrams may arrive out of order, appear duplicated, or go missing without notice. Avoiding the overhead of checking whether every packet actually arrived makes UDP faster and more efficient, at least for applications that do not need guaranteed delivery. Time-sensitive applications often use UDP because dropped packets are preferable to delayed packets. UDP's stateless nature is also useful for servers that answer small queries from huge numbers of clients. Unlike TCP, UDP is compatible with packet broadcast (sending to all on local network) and multicasting (send to all subscribers).

UDP is a minimal message-oriented transport layer protocol that is currently documented in IETF RFC 768. In the Internet protocol suite, UDP provides a very simple interface between a network layer below (e.g., IPv4) and a session layer or application layer above.

UDP provides no guarantees to the upper layer protocol for message delivery and a UDP sender retains no state on UDP messages once sent (for this reason UDP is sometimes called the Unreliable Datagram Protocol). UDP adds only application multiplexing and checksumming of the header and payload. If any kind of reliability for the information transmitted is needed, it must be implemented in upper layers.

+	Bits 0 - 15	16 - 31
0	Source Port	Destination Port
32	Length	Checksum
64	Data	

The UDP header consists of only 4 fields. The use of two of those is optional.

Source port

This field identifies the sending port when meaningful and should be assumed to be the port to reply to if needed. If not used, then it should be zero.

Destination port

This field identifies the destination port and is required.

Length

A 16-bit field that specifies the length in bytes of the entire datagram: header and data. The minimum length is 8 bytes since that's the length of the header. The field size sets a theoretical limit of 65,535 bytes for the data carried by a single UDP datagram. The practical limit for the data length which is imposed by the underlying IPv4 protocol is 65,507 bytes.

Checksum

The 16-bit checksum field is used for error-checking of the header and data.

2.5 Rules

The rules in this project refers to the network intrusion signatures that we check in the data packets. The rule used in this project is similar to the rule description of the snort. The rules used in this IDS are downloaded from the snort site by the admin through web ui. But all the constraints of the rules are not used in this IDS.

The rule description language used in this project is described below:

2.5.1 Rules Headers

The rule header contains the information that defines the who, where, and what of a packet, as well as what to do in the event that a packet with all the attributes indicated in the rule should show up.

Protocols

The next field in a rule is the protocol. There are four protocols that we currently analyzes for suspicious behavior

- TCP, UDP, ICMP, and IP. In the future there may be more, such as ARP, IGRP, GRE, OSPF, RIP, IPX, etc.

Port Numbers

Port numbers may be specified in a number of ways, including any ports, static port definitions, ranges, and by negation. Any ports are a wildcard value, meaning literally any port. Static ports are indicated by a single port number, such as 111 for portmapper, 23 for telnet, or 80 for http, etc. Port ranges are indicated with the range operator :. The range operator may be applied in a number of ways to take on different meanings. Port negation is indicated by using the negation operator !.

2.5.2 Rule Options

Rule options form the heart of intrusion detection engine, combining ease of use with power and flexibility. All rule options are separated from each other using the semicolon

(;) character. Rule option keywords are separated from their arguments with a colon (:) character.

We use three categories of rule options.

general These options provide information about the rule but do not have any affect during detection

payload These options all look for data inside the packet payload and can be inter-related

non-payload These options look for non-payload data

2.5.1.1 General rule Option

msg

The msg rule option tells the engine the message to print along with a log or to an alert.

It is a simple text string that utilizes the \ as an escape character to indicate a discrete character that might otherwise confuse rules parser (such as the semi-colon ; character).

Format

msg: "<message text>";

2.5.1.2 Payload Detection Rule Options

Content

The content keyword is one of the more important features of IDS. It allows the user to set rules that search for specific content in the packet payload and trigger response based on that data. Whenever a content option pattern match is performed, the Boyer-Moore pattern match function is called and the (rather computationally expensive) test is performed against the packet contents.

If data exactly matching the argument data string is contained anywhere within the packet's payload, the test is successful and the remainder of the rule option tests are performed. Be aware that this test is case sensitive. The option data for the content keyword is somewhat complex; it can contain mixed text and binary data. The binary data is generally enclosed within the pipe (|) character and represented as bytecode. Bytecode represents binary data as hexadecimal numbers and is a good shorthand method

for describing complex binary data. There can be multiple content rules can be specified in one rule. This allows rules to be tailored for less false positives.

If the rule is preceded by a !, the alert will be triggered on packets that do not contain this content. This is useful when writing rules that want to alert on packets that do not match a certain pattern

Format

content: [!] "<content string>";

2.5.1.3 Non-Payload Detection Rule Options

fragoffset

The fragoffset keyword allows one to compare the IP fragment offset field against a decimal value. To catch all the first fragments of an IP session, we could use the fragbits keyword and look for the More fragments option in conjunction with a fragoffset of 0.

Format

fragoffset:[<|>]<number>;

ttl

The ttl keyword is used to check the IP time-to-live value. This option keyword was intended for use in the detection of traceroute attempts.

Format

ttl:[[<number>-]><=]<number>;

tos

The tos keyword is used to check the IP TOS field for a specific value.

Format

tos:[!]<number>;

id

The id keyword is used to check the IP ID field for a specific value. Some tools (exploits, scanners and other odd programs) set this field specifically for various purposes, for example, the value 31337 is very popular with some hackers.

Format

id:<number>;

fragbits

The fragbits keyword is used to check if fragmentation and reserved bits are set in the IP header.

The following bits may be checked:

M - More Fragments

D - Don't Fragment

R - Reserved Bit

The following modifiers can be set to change the match criteria:

+ match on the specified bits, plus any others

***** match if any of the specified bits are set

! match if the specified bits are not set

Format

fragbits:[+*!]<[MDR]>;

dsize

The dsize keyword is used to test the packet payload size. This may be used to check for abnormally sized packets. In many cases, it is useful for detecting buffer overflows.

Format

dsize: [<>]<number>[<><number>];

flags

The flags keyword is used to check if specific TCP flag bits are present. The following bits may be checked:

F - FIN (LSB in TCP Flags byte)

S - SYN

R - RST

P - PSH

A - ACK

U - URG

1 - Reserved bit 1 (MSB in TCP Flags byte)

2 - Reserved bit 2

0 - No TCP Flags Set

The following modifiers can be set to change the match criteria:

+ - match on the specified bits, plus any others

***** - match if any of the specified bits are set

! - match if the specified bits are not set

Format

flags:[!|*|+]<FSRPAU120>[,<FSRPAU120>];

seq

The seq keyword is used to check for a specific TCP sequence number.

Format

seq:<number>;

ack

The ack keyword is used to check for a specific TCP acknowledge number.

Format

ack: <number>;

window

The window keyword is used to check for a specific TCP window size.

Format

window:[!]<number>;

itype

The itype keyword is used to check for a specific ICMP type value.

Format

itype:[<|>]<number>[<><number>];

icode

The itype keyword is used to check for a specific ICMP code value.

Format

icode: [<|>]<number>[<><number>];

icmp id

The itype keyword is used to check for a specific ICMP ID value. This is useful because some covert channel programs use static ICMP fields when they communicate.

Format

icmp_id:<number>;

ip proto

The ip proto keyword allows checks against the IP protocol header. For a list of protocols that may be specified by name, see /etc/protocols.

Format

ip_proto:[!|>|<] <name or number>;

sameip

The sameip keyword allows rules to check if the source ip is the same as the destination IP.

Format

sameip;

Chapter 3

METHODOLOGY

3.1 Software Development Technology

We used XP as our software development methodology. XP is actually a deliberate and disciplined approach to software development. XP is successful because it stresses customer satisfaction. The methodology is designed to deliver the software to our needs when it is needed. XP empowers our developers to confidently respond to changing requirements, even late in the life cycle.

This methodology also emphasizes team work. Developers are all part of a team dedicated to delivering quality software. XP implements a simple, yet effective way to enable groupware style development.

XP improves a software project in four essential ways; communication, simplicity, feedback, and courage. We communicated within group, and kept the design simple and clean. We got feedback by testing our software starting on day one. With this foundation XP programmers are able to courageously respond to changing requirements and technology.

XP is different. It is a lot like a jig saw puzzle. There are many small pieces. Individually the pieces make no sense, but when combined together a complete picture can be seen. This is a significant departure from traditional software development methods and ushers in a change in way we program.

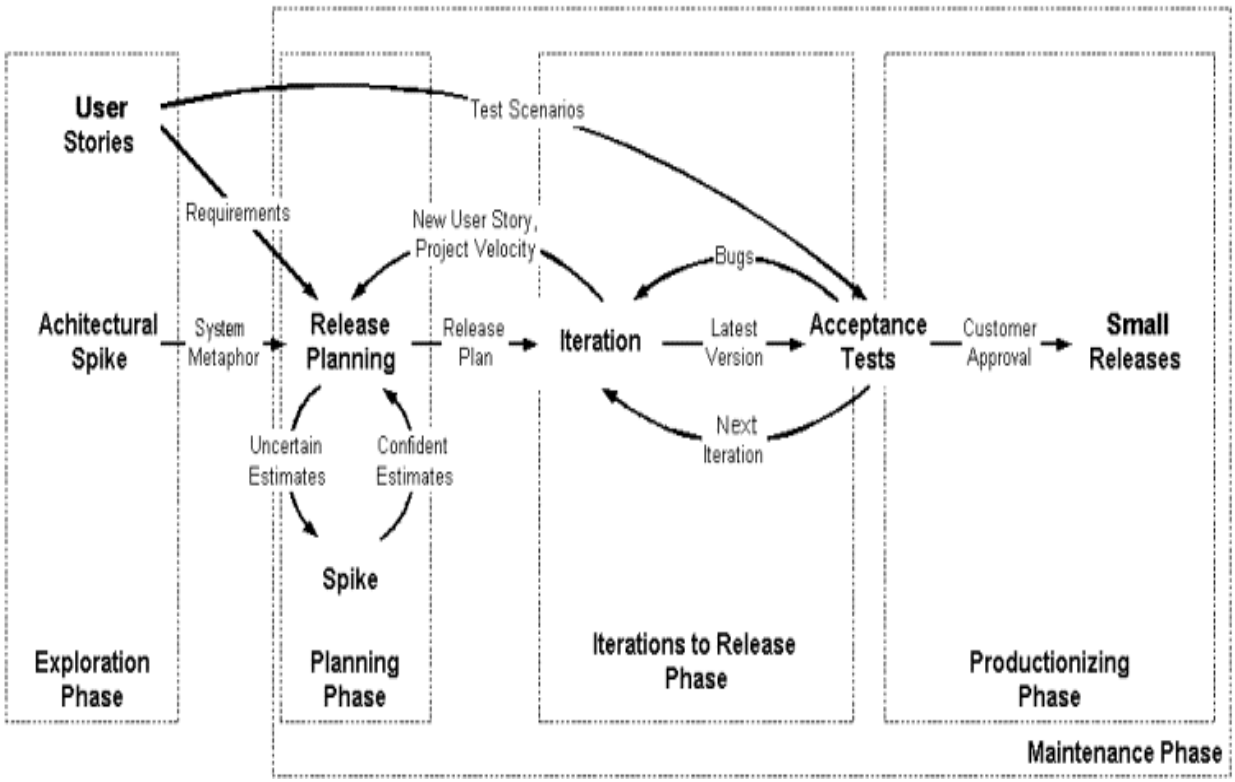

Figure12: Extreme Programming

3.2 Software and tool used

The development tools we've used are:

3.2.1 Java

Java is a programming language originally developed by Sun Microsystems and released in 1995 as a core component of Sun's Java platform. The language derives much of its syntax from C and C++ but has a simpler object model and fewer low-level facilities. Java applications are typically compiled to bytecode which can run on any Java virtual machine (JVM) regardless of computer architecture.

3.2.2 Eclipse

Eclipse is an open-source software framework written primarily in Java. In its default form it is an Integrated Development Environment (IDE) for Java developers, consisting of the Java Development Tools (JDT) and the Eclipse Compiler for Java (ECJ). Users can extend its capabilities by installing plug-ins written for the Eclipse software framework,

such as development toolkits for other programming languages, and can write and contribute their own plug-in modules. Language packs are available for over a dozen languages

3.2.3 Tomcat 6

Apache Tomcat is a web container, or application server developed at the Apache Software Foundation (ASF). Tomcat implements the Java Servlet and the JavaServer Pages (JSP) specifications from Sun Microsystems, providing an environment for Java code to run in cooperation with a web server. It adds tools for configuration and management but can also be configured by editing configuration files that are normally XML-formatted. Tomcat includes its own internal HTTP server.

3.2.4 JSF 1.2

JavaServer Faces (JSF) is a Java-based Web application framework that simplifies the development of user interfaces for Java EE applications. Unlike other traditional request-driven MVC web frameworks, JSF uses a component-based approach. The state of UI components is saved when the client requests a new page and then is restored when the request is returned. Out of the box, JSF uses JavaServer Pages (JSP) for its display technology, but JSF can also accommodate other display technologies (such as XUL).

3.2.5 JADE

Java Agent DEvelopment framework is an open-source Java-based environment to build and host agent-based systems. It follows FIPA standards and contains libraries that provide different levels of communication and control functions that can be used to define the behavior of agents. JADE supports an ACL (Agent Communication Language) message exchange layer as well as a basic ontology that can be extended for specific applications. JADE lacks support for intelligent agent functions but, given that it is Java-based software, it can interact with other systems that do provide specialized functionality (e.g., Jena). JADE models can be distributed across several machines running different operating systems as support for agent location and mobility is also available. A GUI allows users to visualize agents, message exchanges, and the status of

other system components. This simplifies debugging and allows for a more intuitive understanding of the agent platform.

3.2.6 Jpcap

What is Jpcap?

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

- capture raw packets live from the wire.
- save captured packets to an offline file, and read captured packets from an offline file.
- automatically identify packet types and generate corresponding Java objects (for Ethernet, IPv4, Ipv6, ARP/RARP, TCP, UDP, and ICMPv4 packets).
- filter the packets according to user-specified rules before dispatching them to the application.
- send raw packets to the network

Jpcap is based on libpcap/winpcap, and is implemented in C and Java.

Jpcap has been tested on Microsoft Windows (98/2000/XP/Vista), Linux (Fedora, Ubuntu), Mac OS X (Darwin), FreeBSD, and Solaris.

What kind of applications can be developed using Jpcap?

Jpcap can be used to develop many kinds of network applications, including (but not limited to):

- network and protocol analyzers
- network monitors
- traffic loggers

- traffic generators
- user-level bridges and routers
- network intrusion detection systems (NIDS)
- security tools

What Jpcap cannot do?

Jpcap captures and sends packets *independently* from the host protocols (e.g., TCP/IP). This means that Jpcap does not (cannot) block, filter or manipulate the traffic generated by other programs on the same machine: it simply "sniffs" the packets that transit on the wire. Therefore, it does not provide the appropriate support for applications like traffic shapers, QoS schedulers and personal firewalls.

Chapter 4

IMPLEMENTATION

The IDS that we implemented is shown below using the use case diagram. There are mainly two parts of IDS. First one is the core system where all the process takes place while second one is the web based GUI for the easier use for administrator.

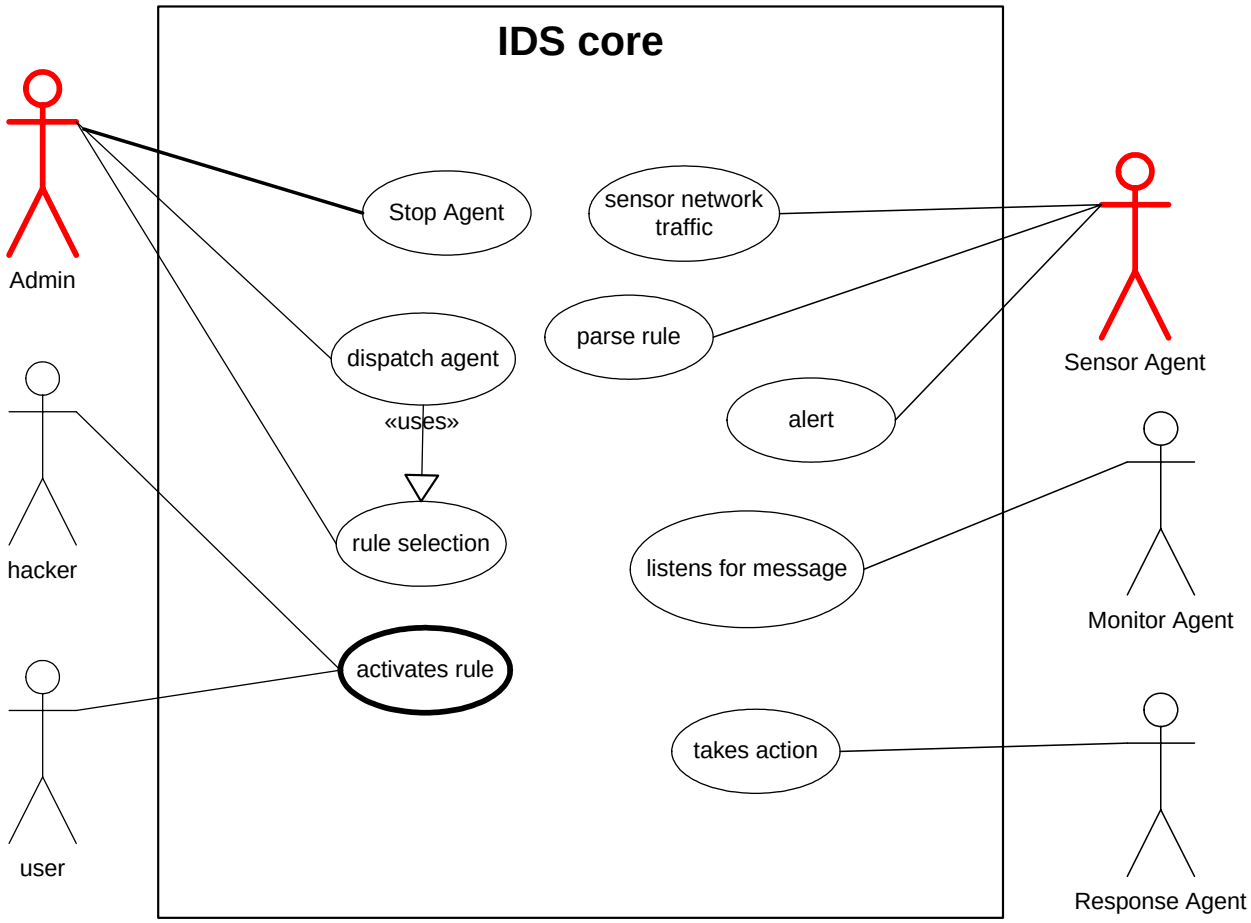

Figure 13: Use case diagram of the IDS core

Above figure shows the overview of the use of the IDS. Admin is responsible for the purpose of dispatching the sensor agent loaded with the required rule set. He can also stop the sensor agent that is dispatched to some host.

Other actors of the IDS are hackers or the internal user who tries to violate the company rules. In case of violation of the law a rule is fired. Then the sensor agent sends a message

to the monitor agent. This agent listens the messages posted by different sensor agent. Then the core dispatches the response agent with the response command that is mentioned in the response table by the admin of the network.

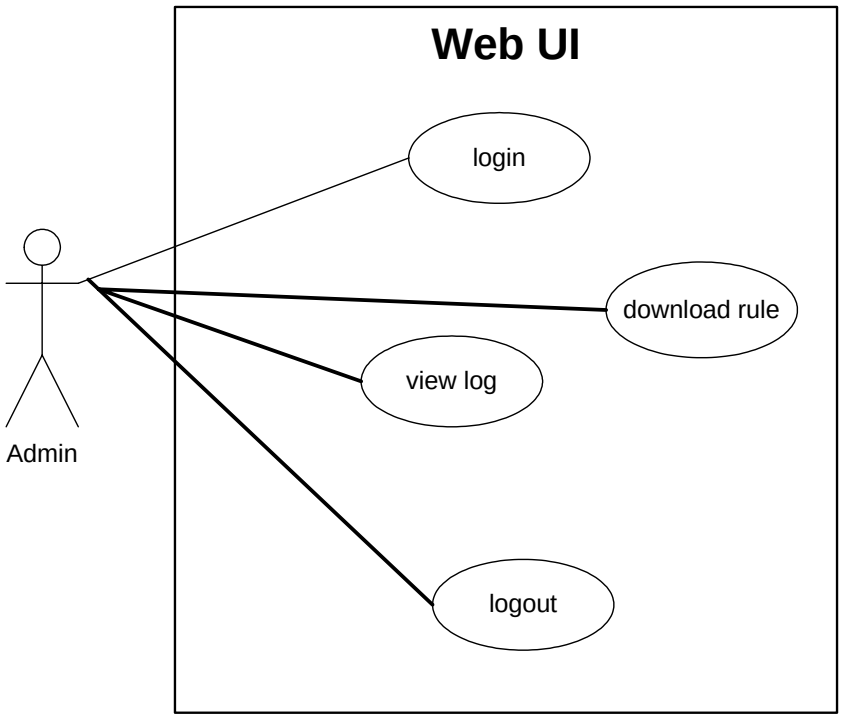

Figure 14: Use case diagram for the web based GUI with IDS

The second subsystem of the IDS is web based gui. This gui provides the basic functionalities for the admin . The user need to log in the system and then he can view the log of the IDS. Not only that he can also download the latest signatures for the IDS.

4.1 Agent Architecture

In this IDS we have made three agents for with different functionalities. The agents are developed using the JADE. The agents of the IDS are compatible to run in multiple platforms. The three agents are:

- Sensor agent: This agent is equipped with the functionalities of capturing data packets, parsing the rules that are given to it and finally analyze the packet with the intrusion signature.

- Monitor Agent: This agent listens to the messages that are send from different Sensor agents. This agent informs the main system about the intrusion according to the message .

- Response Agent: This agent is equipped with the functionalities to take action according to the message of the Sensor agent. The main system put the command from the response table filled by the admin. Then his agent is dispatched to the remote host. In this host it runs this code.

Before starting the agent the remote host where the agent is to be dispatched must be running the JADE container. JADE provides agent name location and communication primitives that have been used to speed up the development of the system. Each agent cell is implemented as an JADE environment also called container which is in turn a Java virtual machine. Each environment executes agent-support classes that provide all basic methods for the functioning of the agent environment. Each container executes multiple concurrent agents using threads and can communicate with other environments through message passing.

A set of JADE behaviours define the functioning of each agent. Behaviors are routines used to group into modules the different methods an agent is able to execute depending on its state and perceptions. The code below shows the Monitor agent with behavior action which is used to listen to the message from other agent.

```
public class AgMonitor extends Agent
{

 protected void setup()
 {
      String localname="monitor";
     AID id=new AID(localname,AID.ISLOCALNAME);
     // register the mobility ontology
```

```
getContentManager().registerOntology(MobilityOntology.getInstance());

//System.out.println("My local-name is "+id.getLocalName());

addBehaviour(new CyclicBehaviour(this)
{
        public void action(){
            -------
            ------
            --
            -----
```

4.2 Network Data Capture

We have used the java library jpcap to capture the data packer which are later used for analyzing the data. We first find out the list of interfaces and then use the eth0 to capture the packet.

```
NetworkInterface[] devices = JpcapCaptor.getDeviceList();

    try{

            JpcapCaptor captor=JpcapCaptor.openDevice(…,..,..,..);
            PacketPrinter pp=new PacketPrinter();
            captor.loopPacket(-1,pp);

            captor.close();
        }
```

The code snippet given above is used for capturing the packet. In the first line we get the device list. Then we select a particular device and capture the packet. The –1 parameter

in loopPacket() function is used for capturing the for undefined no of packets. The other parameter is the object of the PacketPrinter class which need to implement the PacketReciever interface which is shown in the code snippet below.

```
public class PacketPrinter implements PacketReceiver {


    public void receivePacket(Packet packet) {
        …….
        ……
        …
        …….
        ..
```

The recievePacket() method of this class is called every time the Jpcap captures the packet. We manipulate this captured packet.

4.3 Parsing Rules

The rules used in this IDS need to be parsed and checked with the particular parameter of the packet for checking if there is some intrusion. The total no of reserved word/characters that is present in this rule description language is 37. We have used the java regex class to parse these rules syntax.

What Are Regular Expressions?

Regular expressions are a way to describe a set of strings based on common characteristics shared by each string in the set. They can be used to search, edit, or manipulate text and data. We must learn a specific syntax to create regular expressions — one that goes beyond the normal syntax of the Java programming language. Regular

expressions vary in complexity, but once we understand the basics of how they're constructed, we'll be able to decipher (or create) any regular expression.

This section discusses the regular expression syntax supported by the java.util.regex API and presents several working examples to illustrate how the various objects interact. In the world of regular expressions, there are many different flavors to choose from, such as grep, Perl, Tcl, Python, PHP, and awk. The regular expression syntax in the java.util.regex API is most similar to that found in Perl.

How Are Regular Expressions Represented in This Package?

The java.util.regex package primarily consists of three classes: Pattern, Matcher, and PatternSyntaxException.

- A Pattern object is a compiled representation of a regular expression. The Pattern class provides no public constructors. To create a pattern, we must first invoke one of its public static compile methods, which will then return a Pattern object. These methods accept a regular expression as the first argument;
- A Matcher object is the engine that interprets the pattern and performs match operations against an input string. Like the Pattern class, Matcher defines no public constructors. We obtain a Matcher object by invoking the matcher method on a Pattern object.
- A PatternSyntaxException object is an unchecked exception that indicates a syntax error in a regular expression pattern.

String Literals

The most basic form of pattern matching supported by this API is the match of a string literal. For example, if the regular expression is foo and the input string is foo, the match will succeed because the strings are identical. Try this out with the test harness:

Enter your regex: foo

Enter input string to search: foo

I found the text "foo" starting at index 0 and ending at index 3.

This match was a success. Note that while the input string is 3 characters long, the start index is 0 and the end index is 3. By convention, ranges are inclusive of the beginning index and exclusive of the end index, as shown in the following figure:

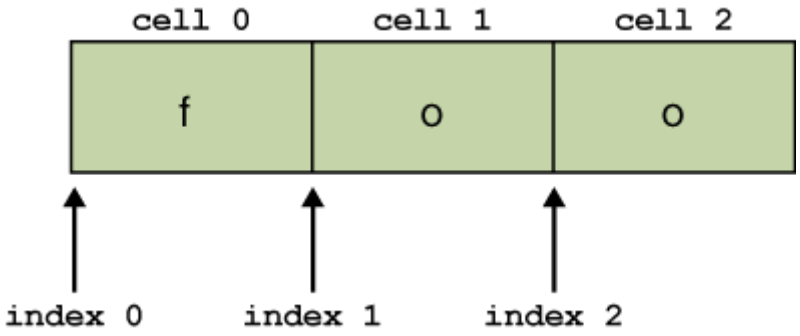

Figure15: Pattern matching of the string literals.

Metacharacters

This API also supports a number of special characters that affect the way a pattern is matched. Change the regular expression to cat. and the input string to cats. The output will appear as follows:

Enter your regex: cat.

Enter input string to search: cats

I found the text "cats" starting at index 0 and ending at index 4.

The match still succeeds, even though the dot "." is not present in the input string. It succeeds because the dot is a *metacharacter* — a character with special meaning interpreted by the matcher. The metacharacter "." means "any character" which is why the match succeeds in this example.

The metacharacters supported by this API are: ([{\^-$|]})?*+.

There are two ways to force a metacharacter to be treated as an ordinary character:

- precede the metacharacter with a backslash, or
- enclose it within \Q (which starts the quote) and \E (which ends it).

When using this technique, the \Q and \E can be placed at any location within the expression, provided that the \Q comes first.

Character Classes

The character classes are clearly seen from the formatted text below.

[abc]	a, b, or c (simple class)
[^abc]	Any character except a, b, or c (negation)
[a-zA-Z]	a through z, or A through Z, inclusive (range)
[a-d[m-p]]	a through d, or m through p: [a-dm-p] (union)
[a-z&&[def]]	d, e, or f (intersection)
[a-z&&[^bc]]	a through z, except for b and c: [ad-z] (subtraction)
[a-z&&[^m-p]]	a through z, and not m through p: [a-lq-z] (subtraction)

Predefined Character Classes

The Pattern API contains a number of useful *predefined character classes*, which offer convenient shorthands for commonly used regular expressions:

.	Any character (may or may not match line terminators)

\d	A digit: [0-9]
\D	A non-digit: [^0-9]
\s	A whitespace character: [\t\n\x0B\f\r]
\S	A non-whitespace character: [^\s]
\w	A word character: [a-zA-Z_0-9]
\W	A non-word character: [^\w]

Quantifiers

Quantifiers allow us to specify the number of occurrences to match against. For convenience, the three sections of the Pattern API specification describing greedy, reluctant, and possessive quantifiers are presented below. At first glance it may appear that the quantifiers X?, X?? and X?+ do exactly the same thing, since they all promise to match "X, once or not at all".

Quantifiers can only attach to one character at a time, so the regular expression "abc+" would mean "a, followed by b, followed by c one or more times". It would not mean "abc" one or more times. However, quantifiers can also attach to Character Classes and Capturing Groups, such as [abc]+ (a or b or c, one or more times) or (abc)+ (the group "abc", one or more times).

Capturing Groups

Capturing groups are a way to treat multiple characters as a single unit. They are created by placing the characters to be grouped inside a set of parentheses. For example, the regular expression (dog) creates a single group containing the letters "d" "o" and "g". The portion of the input string that matches the capturing group will be saved in memory for later recall via backreferences.

Boundary Matchers

We can make pattern matches more precise by specifying such information with *boundary matchers*. For example, maybe we're interested in finding a particular word, but only if it appears at the beginning or end of a line. Or maybe we want to know if the match is taking place on a word boundary, or at the end of the previous match.

The following table lists and explains all the boundary matchers.

^	The beginning of a line
$	The end of a line
\b	A word boundary
\B	A non-word boundary
\A	The beginning of the input
\G	The end of the previous match
\Z	The end of the input but for the final terminator, if any
\z	The end of the input

4.4 Matching Intrusion Signature

In this section we discuss the process of matching the parsed rules constraints with the packet parameters. The sequence diagram given below gives the basic idea of how the pattern matching engine is implemented.

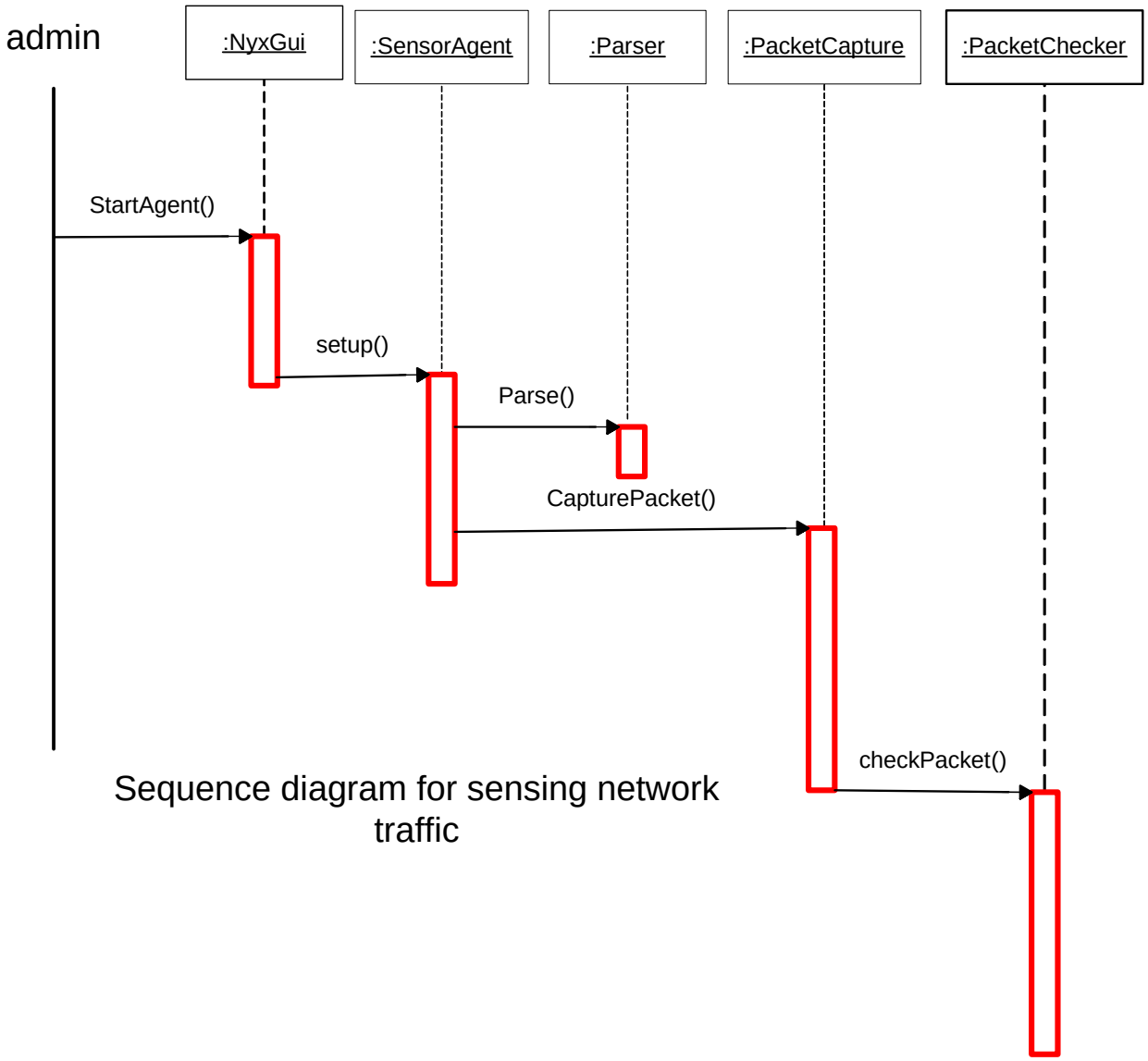

Figure 16: Sequence diagram 1

When the admin selects a particular host to start the agent, NyxGui class calls the setup() method of the agent. After agent is dispatched to the remote host it parses the rules using the Parser class. Meanwhile the sensor agent also captures the packet from the network. This captured packet is then analyzed with the rules constraints to check the anomaly in the network traffic.

4.5Response Mechanism

The response mechanism of IDS is simple yet effective. In the system core the response table is maintained by the admin for response to the particular attack.

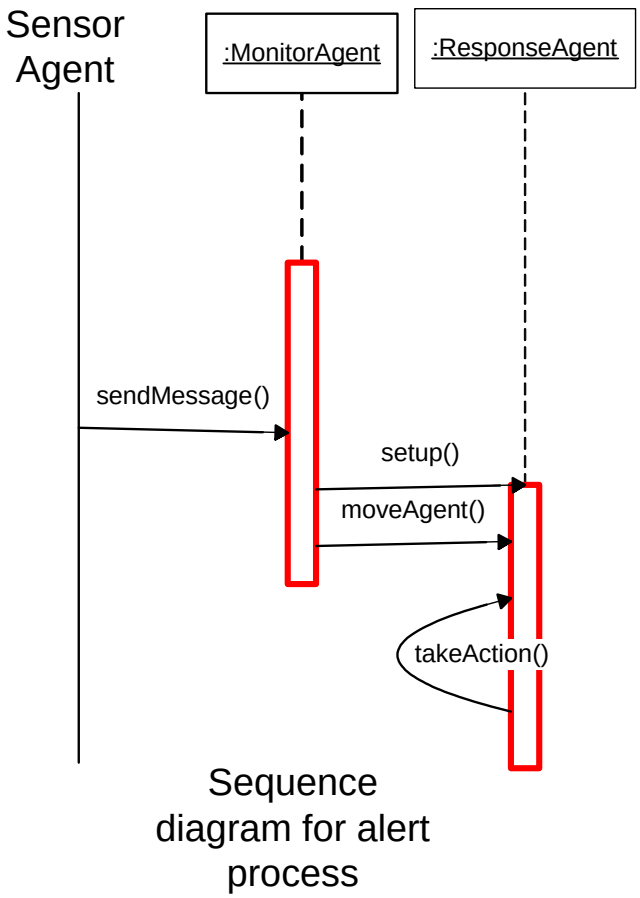

Figure17: Sequence diagram 2

When sensor agent senses a anomalous packet, it sends message to the Monitor Agent. Then monitor agent start a response agents add the response command to it from the response table and finally dispatches the response agent to the remote computer where it runs the particular command it is supposed to carry out.

4.6 Web UI

The IDS is equipped with the web based GUI for carrying out certain functionalities such as viewing log and downloading the latest rules. The sequence diagram below explains the sequence in which the methods are called of particular object to while performing certain tasks.

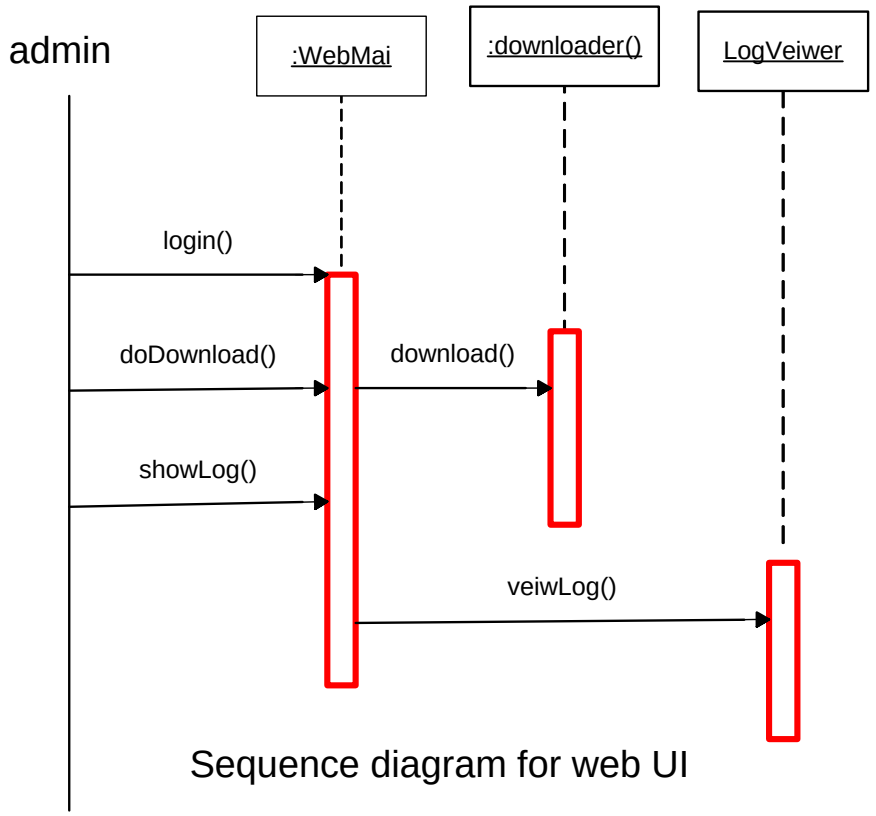

Figure 18: Sequence diagram 3

The admin log into the system and after logging in he can carry out the features that are provided by the web UI . If he wants to download the latest rules he will select the rule downloading option and the rules will be downloaded to the directory from which the core will now have access to the latest rules. The admin also has authority to view the log which is posted in case of intrusion in the host computers. He can then take the required action on strengthening the security, punishing the defaulter if he is an insider or provide the log to the police, which will help in the forensic, to find out the culprit.

4.7 Screen Shots

Screen shots show the real output of the system. We are proud to show the screen shots by which our real work is displayed. We have different facilities for the administrator to gain directly from the GUI. The administrator can login with his user name and password to access those different facilities. These facilities are such as viewing log, update rules,

setting the command for different type of intrusion activities, useful links and several general information about the Intrusion Detection System.

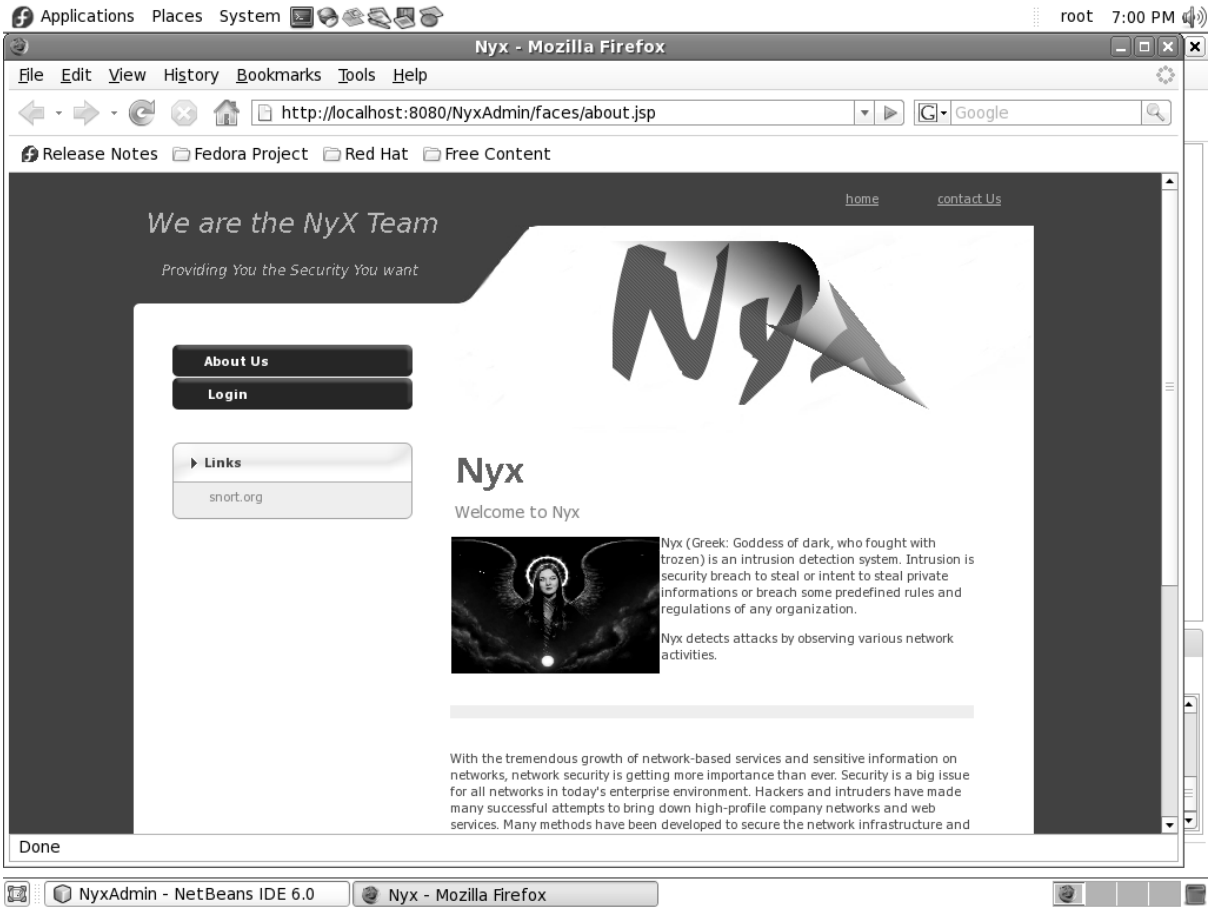

Figure 19: Screen shot 1showing information about Nyx.

Figure 20: Showing Information about the admin login page.

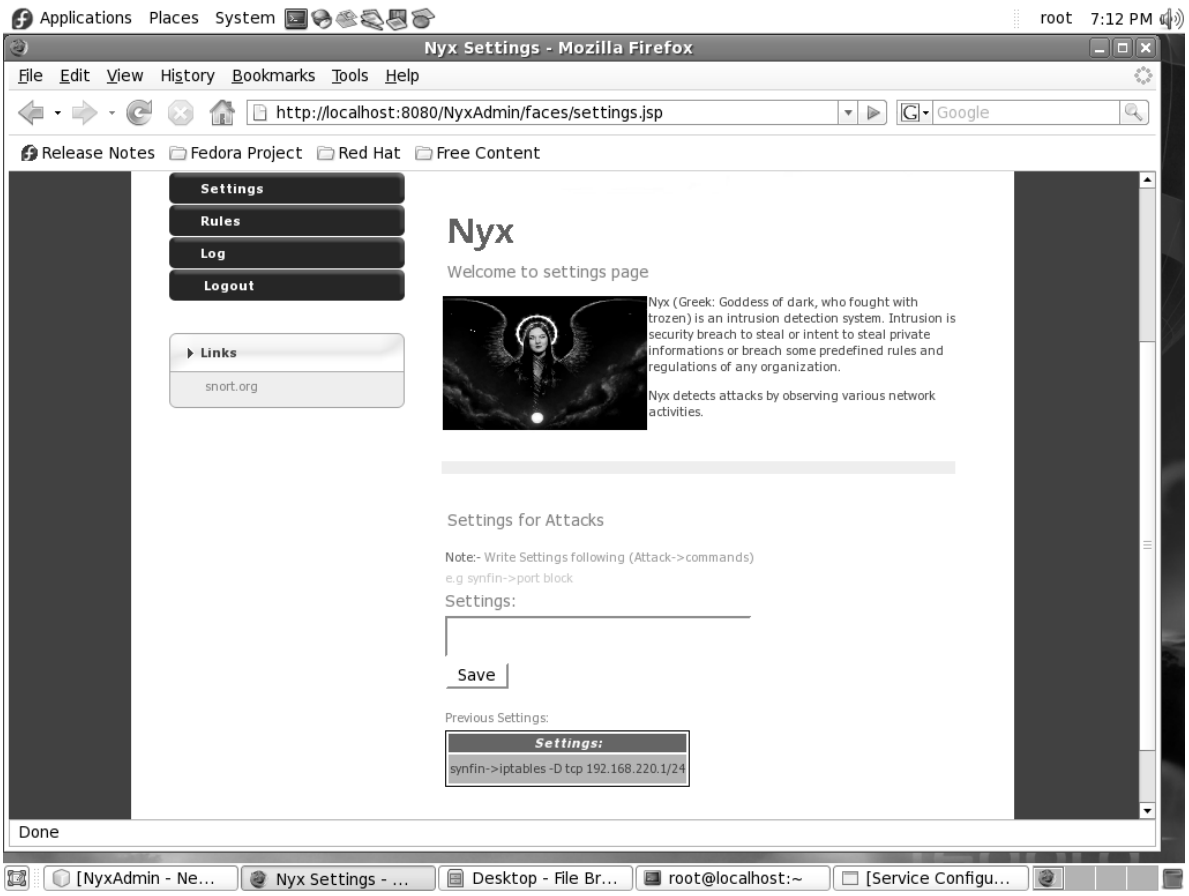

Figure 21: Administrator making settings for different attacks.

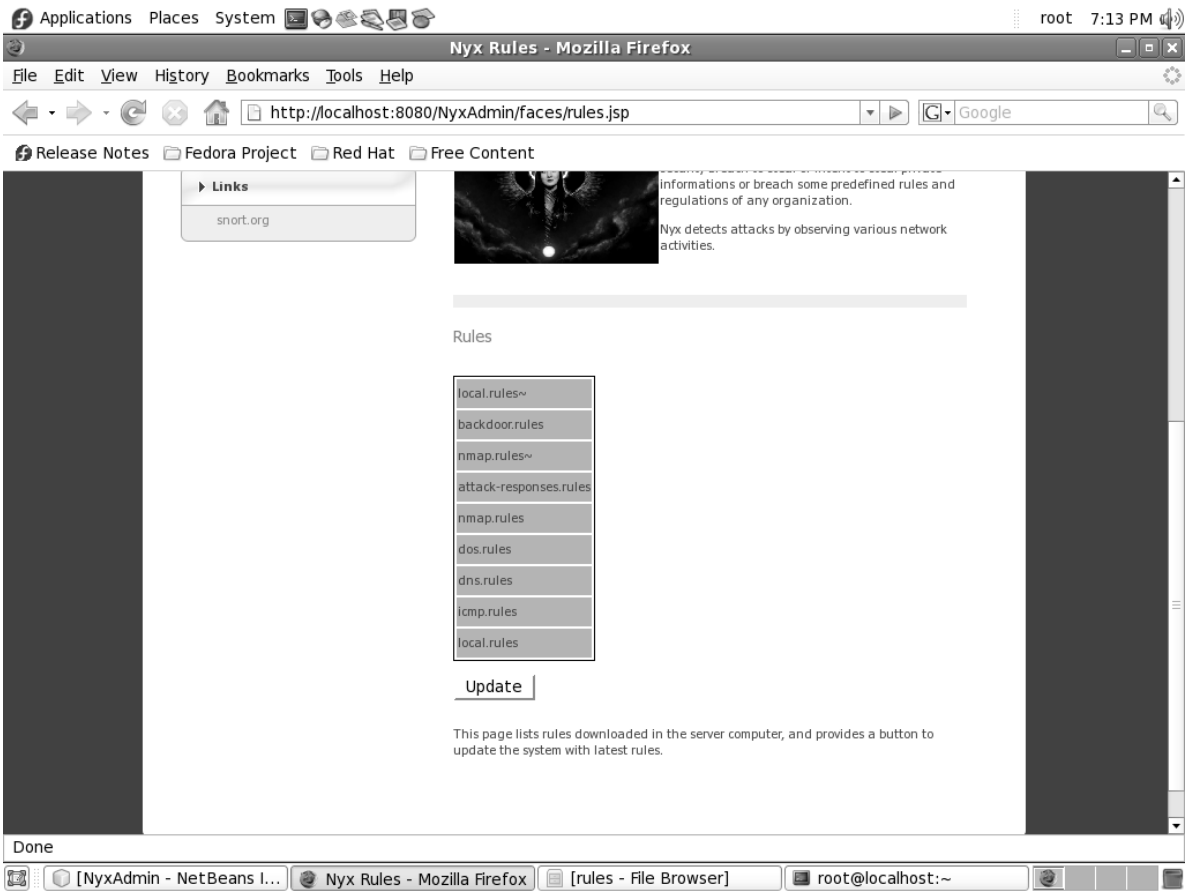

Figure 22: Place for which administrator can update different rules in a click.

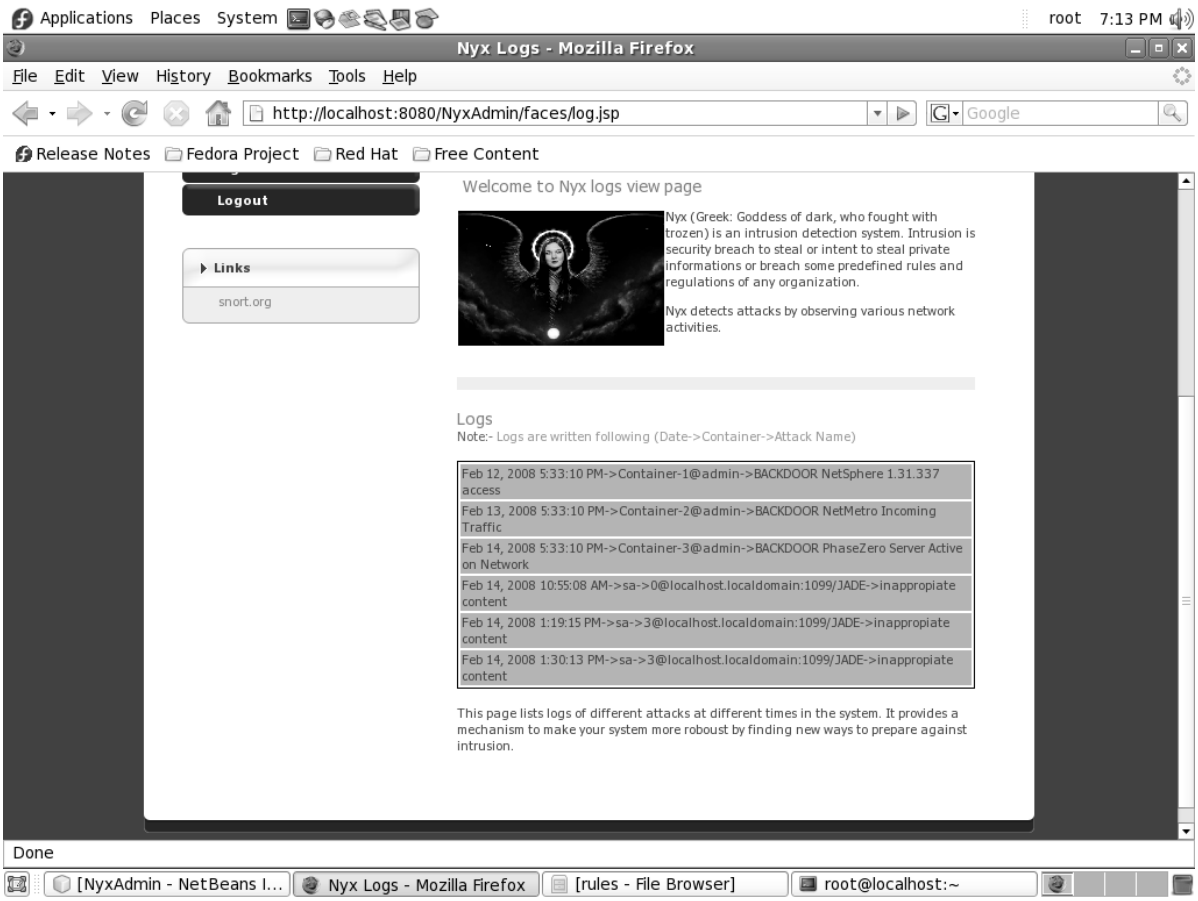

Figure 23: GUI to display the logs recoded by the system.

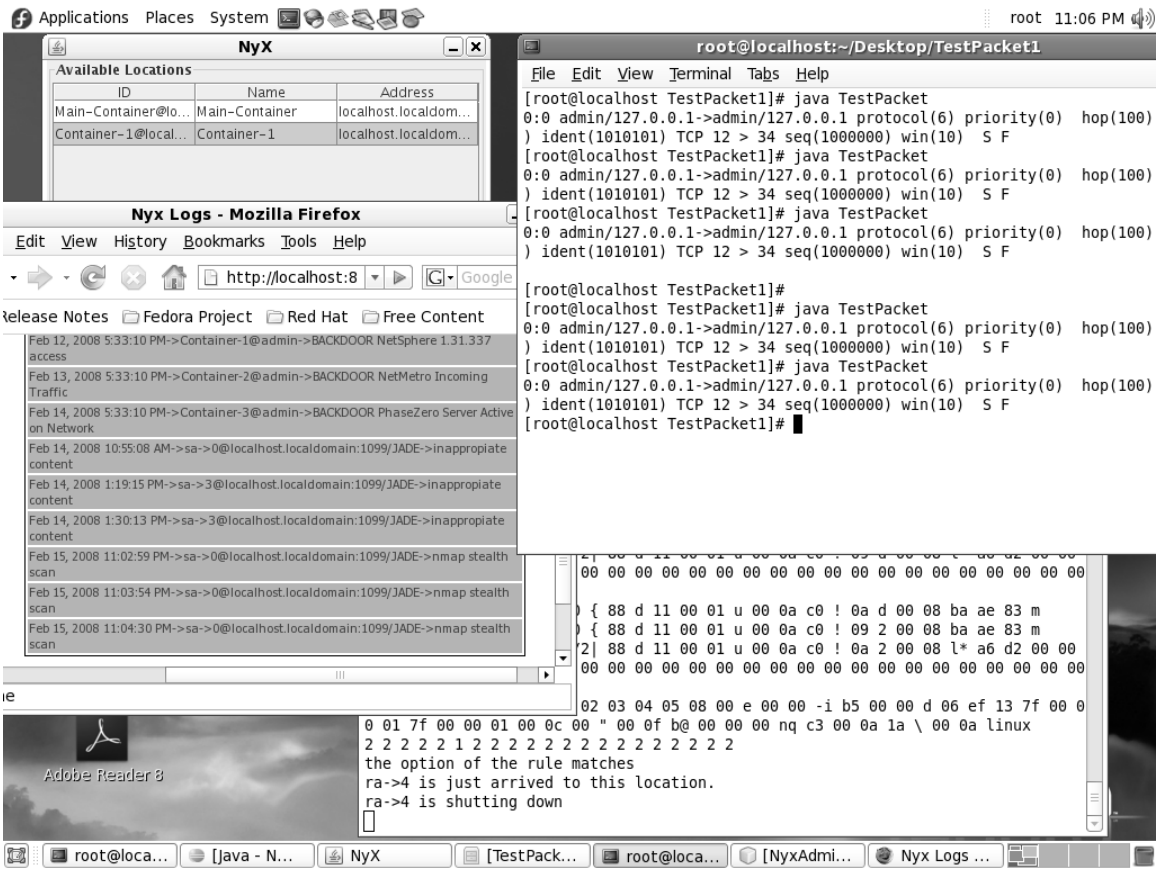

Figure 24: Main panel along with the agent movement terminal.

These screen shots reveals our work directly. Our system can checks all the signatures described above. As our system can check the content of the packet we can create the firewall like great firewall of China “Tian'anmen Square scandal”. These type of firewall can also be created relating to explicit words which may be suitable for the schools/collages and religious society.

Chapter 5

Conclusion, Recommendation and Future Enhancement:

5.1 Conclusion

In the diagram below, we have presented general graphical overview of different parameters that becomes constraints for efficiency of intrusion detection system. We take snort into account because it can be regarded as ideal intrusion detection system. Snort is more accurate than our IDS as our system(IDS) don't support the fragmentation efficiently. IDS has efficient computing power as it uses distributed agents computed in different computers where as snort is server oriented. In case of snort there is possibility of bottleneck but our system avoids that. So speed of comparatively faster in IDS then that of snort. Snort has command line configuration mode where as IDS has GUI for the configuration. So IDS is better in the ease of use. There are above 50 people directly involved in snort project. Snort has come to this stage after 7 years of development cycle.

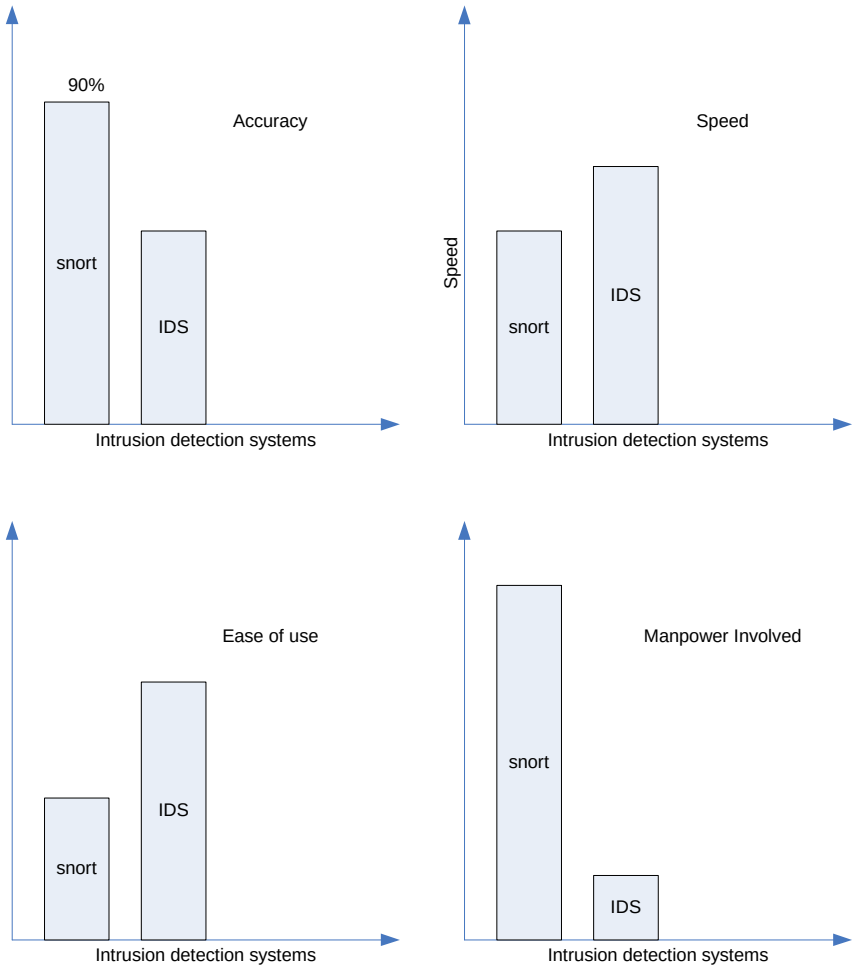

Figure 25: Comparison between IDS and Snort.

After completing this project we made a signature based Intrusion Detection System. We are able to do teamwork and knew the way to task dividing and cooperating in the task. Successful work not only made us feel proud but we also became good companions. In this way we completed our project successfully.

5.2 Future work Recommendation and Further Enhancement:

We had a nice experience for doing research and working on Intrusion Detection System. It is the field which is still to be developed. Our work was output based on materials published in different research papers. This detection system is based on signature and use of agents. This Intrusion Detection System is network based intrusion detection system (NIDS).For better efficiency we can shift to hybrid based intrusion detection system (HIDS). HIDS has both features of analyzing packets as well as log.

We can increase the performance of IDS by using preprocessor to support fragmentation and analyzing different application layer protocols. From the content we can extract necessary information and this will help in further enhancing the system.

Our GUI can be more interactive and should looks like commercially attractive. So it must be further worked to make so.

We will leave this work open for the continuity in our college and community. In foreign universities there are separate research departments for intrusion detection system. Intrusion Detection System is completely challenging and demanding project for this age of information. There is no ending boundary for this project. As day by day new intruder are seen. The only thing is to increase the efficiency of the system to detect new intruder intelligently. At first we should be able to have resources for huge data files of logging traffic. Then there should be data mining techniques applied to trace or acquire needed anomalies and abnormalities. We would like to recommend to develop own rules for which this project should be continuously enhanced day by day. So that, system can be developed into one that will challenge the monopoly of snort.

List of Abbreviations

IDS	Intrusion Detection System
NIDS	Network Based Intrusion Detection System
HIDS	Host Based Intrusion Detection System
IP	Internet Protocol
TCP	Transmission Control Protocol
IT	Information Technology
MAC	Media Access Control
DHCP	Dynamic Host Configuration Protocol
CIDR	Classless inter-domain routing
DNS	Domain Naming System
AS	Autonomous System
ICMP	Internet Control Message Protocol
JVM	Java Virtual Machine
XP	Extreme Programming
JSF	Java Server Faces
JADE	Java Agent DEvelopment framework
GUI	Graphical User Interface

List of Figures

List of Tables

BIBLIOGRAPHY

- F. B. Cohen, “Simulating Cyber Attacks, Defenses, and Consequences,” Available at http://all.net/journal/ntb/simulate/simulate.html, May 13, 1999.
- CERT. Statistics of the Computer Emergency Response Team Coordination Center, CERT–CC, Carnegie Mellon University, 2004. URL: http://www.cert.org/stats/ (on-line resource).
- Salvador Mandujano, “A Multiagent Approach to Outbound Intrusion Detection”, December – 2004.
- Jai Sundar Balasubramaniyan, Jose Omar Garcia-Fernandez, David Isacoff, Eugene Spafford, Diego Zamboni, “An Architecture for Intrusion Detection using Autonomous Agents”, Purdue University.
- Curtis A. Carver, Jr., John M.D. Hill, John R. Surdu Member, IEEE, and Udo W. Pooch, Senior Member, IEEE, “A Methodology for Using Intelligent Agents to provide Automated Intrusion Response”, Proceedings of the 2000 IEEE Workshop on Information Assurance and Security, United States Military Academy, West Point, NY, 6-7 June, 2000.
- Wenke Lee, Salvatore J. Stolfo, “Data Mining Approaches for Intrusion Detection”, Computer Science Department, Columbia University.
- Jiong Zhang and Mohammad Zulkernine, “Network Intrusion Detection using Random Forests”, School of Computing, Queen’s University, Kingston, Ontario, Canada.
- Jake Ryan, Meng-Jang Lin, Risto Miikkulainen, “Intrusion Detection with Neural Networks”, Department of Electrical and Computer Engineering, The University of Texas at Austin, Austin, TX 78712
- http://jade.tilab.com/

- Network intrusion Detection,third edition,Northcutt and Novak
- Snort_manual from www.snort.org
- Srinivas Mukkamala, Andrew Sung and Ajith Abraham* “Designing Intrusion Detection Systems: Architectures,Challenges and Perspectives” Department of Computer Science, New Mexico Tech, USA, *Department of Computer Science, Oklahoma State University, USA
- Salvador Mandujano, Arturo Galv´an, and Juan A. Nolazco “**An Ontology-based Multiagent Architecture for Outbound Intrusion Detection”**