NETWORK FIREWALL SYSTEM

Submitted by
Smily Bansal(101334)

Under the guidance of
Ms. Komal Mahajan

MAY-2014

Submitted in partial fulfillment of the Degree of Bachelor of Technology

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
JAYPEE UNIVERSITY OF INFORMATION TECHNOLOGY-WAKNAGHAT

CERTIFICATE

This is to certify that the work titled "**Network Firewall *System*** : *A Centralized Network Security Application*" submitted by Smily Bansal(101334) in partial fulfillment for the award of degree of BTech in the Discipline of Jaypee University of Information Technology, Waknaghat, has been carried out under my supervision. This work has not been submitted partially or wholly to any other University or Institute for the award of this or any other degree or diploma.

Signature of Supervisor	: ____________________
Name of Supervisor	: Ms. Komal Mahajan
Designation	: Lecturer, Department Of CSE , JUIT Waknaghat.
Date	: ____________________

ACKNOWLEDGMENT

I would most of all like to thank my supervisor Ms. Komal Mahajan (Lec, CSE) for the invaluable advice and positive encouragement they have provided me throughout the course of this project. We would like to express our deep and sincere gratitude to my project guide for being a staunch supporter and motivator of this project. Right from the inception of this project work, my supervisor guided me till the very end in the true sense of the word. She always came up with innovative ways and creative terms thus also helping me to instill and enhance the quality of creative thinking within myself.

Sincere thanks to **Brig(retd.) S.P Gharera , HOD, CSE & IT Department**, for being cooperative to the students of the department and providing relevant guidance in their endeavors.

I would also like to express my gratitude to this alma mater **JUIT, Waknaghat** for providing proper resources as and when required such as an all time internet facility and other resources.

Hence without giving a warm thanks to all of them who made this project work a reality my work would be incomplete.

Smily Bansal

Date : May 14,2014

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF ABBREVIATIONS USED

ADSL	Asymmetric Digital Subscriber Line
ARP	Address Resolution Protocol
ARPANET	Advanced Research Projects Agency Network
ATM	Asynchronous Transfer Mode
DHCP	Dynamic Host Configuration Protocol
DPC	Deep Packet Capture
DPI	Deep Packet Inspection
FDDI	Fiber Distributed Data Interface
FTP	File Transfer Protocol
GRE	Generic Routing Encapsulation
HTTP	Hypertext Transfer Protocol
ICMP	Internet Control Message Protocol
IEEE	Institute Of Electrical And Electronics Engineers
IETF	Internet Engineering Task Force
IMAP	Internet Message Access Protocol
IONL	Internal Organization of the Network Layer
IP	Internet Protocol
ISDN	Integrated Services Digital Network
ISP	Internet Service Provider
LAN	Local Area Network
MAC	Medium Access Control
MIME	Multipurpose Internet Mail Extensions
NAC	Net Access Corporation
OSI	Open system interconnection
OSIRMMF	OSI Reference Model Management Framework
PING	Partimage Is Not Ghost
POP3	Post Office Protocol 3
PPP	Point-to-Point Protocol
RFC	Request for Comment
RPC	Remote Procedure Call
SMB	Server Message Blocks
SMTP	Simple Mail Transfer Protocol
SOAP	Simple Object Access Protocol
SSH	Secure Shell Remote Protocol
TCP	Transmission Control Protocol
Telnet	Telnet Remote Protocol
TTL	Time to Live
UDP	User Datagram Protocol

Abstract

Packet is a unit of data that is routed between an origin and destination on any packet - switched network. When a file is sent over the network the tcp layer of tcp/ip protocol divides it into chunks for efficient routing. A packet sniffer program monitors the network traffic by grabbing information traveling over a network, thereby saving the transmitted files. The application captures every packet on the wire to display important information such as a list of packets and network connections and other vital statistics. Packet capturing is required when we need to see what client and server are actually saying to each other or when we need to analyze the type of traffic on network. It is also required for the understanding of network protocols to be used effectively. These functionalities are further utilized for the purposes of packet filtering by the Firewall application. The captured packets are checked as per the rule-set defined within the Firewall application. The rule-set differentiates the packets to be allowed to enter into the LAN and the ones which are to be blocked from entering into the LAN. Therefore, the security of the personal network is managed by this application and it decides upon whether the incoming and outgoing packets are malicious or not and thereby permits and deny the packets accordingly.

CHAPTER1- INTRODUCTION

1.1 PACKET FILTERING FIREWALL

A packet filtering firewall is computer software or computer hardware that filters all network traffic between your computer, home network, or company network and the Internet. As data streams flow across the network, the concerned application captures each packet and eventually decodes and analyzes its content according to the appropriate RFC or other specifications. After the above steps it further analyzes the packets according to the rule-set and hence decides whether the packet shall be permitted or denied.

All Internet traffic travels in the form of **packets**. A packet is a quantity of data of limited size, kept small for easy handling. When larger amounts of continuous data must be sent, it is broken up into numbered packets for transmission and reassembled at the receiving end. All your file downloads, Web page retrievals, emails -- all these Internet communications **always occur in packets**.

IP packets are composed of a header and payload. The IPv4 packet header consists of:

1. 4 bits that contain the *version*, that specifies if it's an IPv4 or IPv6 packet,
2. 4 bits that contain the *Internet Header Length* which is the length of the header in multiples of 4 bytes. Ex. 5 is equal to 20 bytes.
3. 8 bits that contain the *Type of Service*, also referred to as Quality of Service (QoS), which describes what priority the packet should have,
4. 16 bits that contain the *length* of the packet in bytes,
5. 16 bits that contain an *identification tag* to help reconstruct the packet from several fragments,
6. 3 bits that contain a zero, a flag that says whether the packet is allowed to be *fragmented* or not (DF: Don't fragment), and a flag to state whether more fragments of a packet follow (MF: More Fragments)
7. 13 bits that contain the *fragment offset*, a field to identify which fragment this packet is attached to,
8. 8 bits that contain the *Time to live* (TTL) which is the number of hops (router, computer or device along a network) the packet is allowed to pass before it dies (for

example, a packet with a TTL of 16 will be allowed to go across 16 routers to get to its destination before it is discarded),

9. 8 bits that contain the *protocol* (TCP, UDP, ICMP, etc...)
10. 16 bits that contain the *Header Checksum,* a number used in error detection,
11. 32 bits that contain the *source IP address,*
12. 32 bits that contain the *destination address*

The above fields in the IP packet header serve as the basis of the packet filtering function implementation. As each packet passes through the firewall, it is examined and information contained in the header is compared to a pre-configured set of rules or filters. An allow or deny decision is made based on the results of the comparison. Each packet is examined individually without regard to other packets that are part of the same connection.

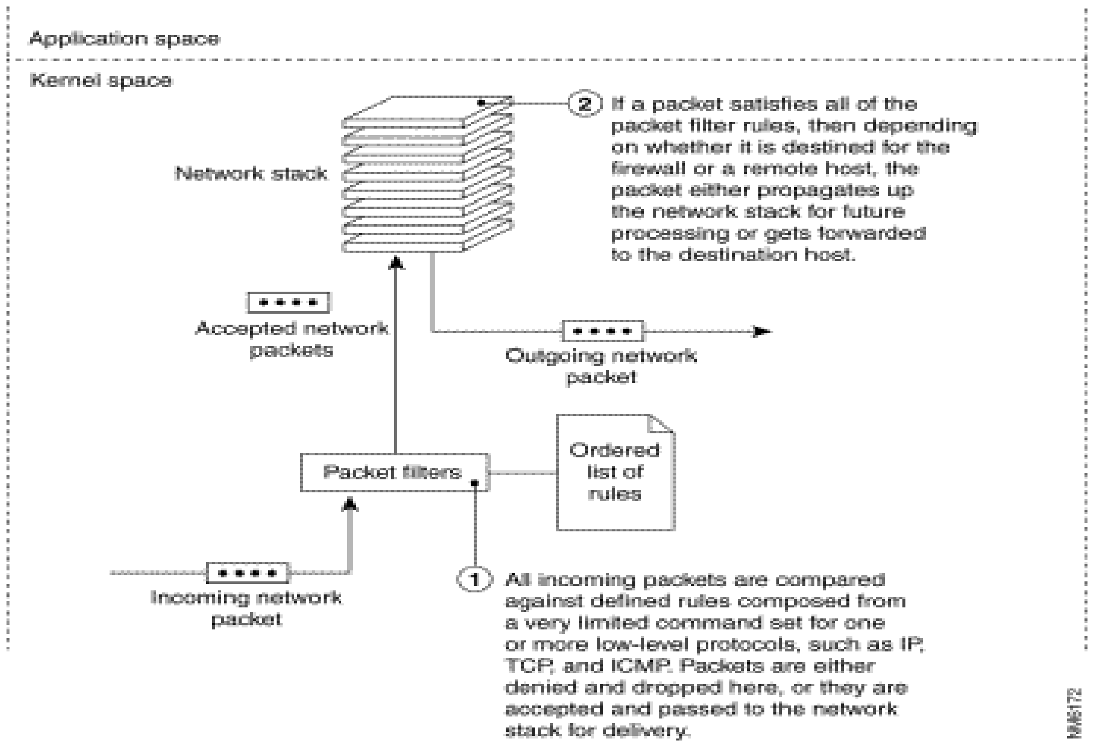
Packet filtering rules or filters can be configured to allow or deny traffic based on one or more of the following variables:

- Source IP address
- Destination IP address
- Protocol type (TCP/UDP)
- Source port
- Destination port

1.1.1 Implementation with respect to the developed application "Firewall"

The developed firewall application code was written in Java using a Java Class Utility known as "**Jpcap**". The implemented firewall permits or denies the accesses to Internet services with packet filtering rule set (PFR). To validate packets that pass the firewall, the firewall checks the PFR . A windows utility known as "**Winpcap**" has been utilized for the packet capturing as well as the relaying process on the network. Furthermore another windows platform utility for packet filtering known as "Ipseccmd.exe" has been used. These utilities are the standard utilities as recommended by the Operating system developer in order to enable kernel programming on the windows platform through the API(Application Platform Interface). Java programming language has been used to develop the application using the "Netbeans7.3.1" Java code developing application. This firewall application developed uses the various above named utilities to perform the packet sniffing and filtering process along with the detailed report on the breaching by the various packets defined according to the rule-set established by the administrator entity. This application also acts as a Network Monitor which allows the end user as well as the administrator to get the details of a particular network which may include the various users, protocols used by the various communications, IP addresses and MAC addresses of the source and the destination entities.

1.1.2 Design Modules


Application space
Kernel space
Network stack
2 If a packet satisfies all of the packet filter rules, then depending on whether it is destined for the firewall or a remote host, the packet either propagates up the network stack for future processing or gets forwarded to the destination host.
Accepted network packets
Outgoing network packet
Packet filters
Ordered list of rules
Incoming network packet
1 All incoming packets are compared against defined rules composed from a very limited command set for one or more low-level protocols, such as IP, TCP, and ICMP. Packets are either denied and dropped here, or they are accepted and passed to the network stack for delivery.
NM6172

- Packet Capture Module
- Packet Filtering Module
- User Interface Module

Packet Capture Module:

This module will integrate with library winpcap and provide a method to investigate the packet. Winpcap is a tool that allows applications to capture and transmit network packets. The basic function of this particular module is to capture raw packets from the network and further relay them to the concerned application for further review with respect to the packet filtering rule-set. So the packets are captured from the network and before they are processed by the operating system for protocol processing purposes, they are passed onto the Java application for packet filtering operations.

Packet filtering Module:

This module will parse the packet header and identify the details. Packet filtering is the selective passing or blocking of data packets as they pass through a network interface. The criteria that packet filtering uses when inspecting packets are based on the Layer 3 (IPv4 and IPv6) and Layer 4 (TCP, UDP, ICMP, and ICMPv6) headers. The most often used criteria are source and destination address, source and destination port, and protocol.

Filter rules specify the criteria that a packet must match and the resulting action, either block or pass, that is taken when a match is found. Filter rules are evaluated in sequential order, first to last. Unless the packet matches a rule containing the quick keyword, the packet will be evaluated against *all* filter rules before the final action is taken. The last rule to match is the "winner" and will dictate what action to take on the packet. There is an implicit pass all at the beginning of a filtering rule-set meaning that if a packet does not match any filter rule the resulting action will be passed.

User Interface Module:

- This module will have the user interface and method to trigger actions based on user request. It will use the other two modules to accomplish the triggered action.
- This Module will be implemented in java using swing and awt components

1.1.3 Capabilities

On wired broadcast LANs, we can capture traffic on all parts of the network from a single machine within the network; however, there are some methods to avoid traffic narrowing by switches to gain access to traffic from other systems on the network (e.g. ARP spoofing). For the purpose of network security all data packets in a LAN can be monitored by using a network switch with a monitoring port, whose purpose is to mirror all packets passing through all the ports of the switch. Further, a packet filtering application can provide network security by providing the functionality to block malicious IP packets. However if the computer is connected to a switch port the analyzer will be unable to read the data owing to the intrinsic nature of switched networks. In this case a shadow port is created so that the analyzer can capture data. In order to capture traffic on wired broadcasts ,other than unicast traffic sent to the machine running the analyzer, multicast traffic sent to a multicast group to which that machine is listening, and broadcast traffic. After the completion of packet capture process the application works upon the packet filtering process. The packets are analyzed according to the source address, destination address and the port address. This information is contained in the packet header and this header information is utilized for packet filtering process. Therefore the network is secured from any attack by malicious addresses present outside the network thereby reducing the vulnerability of the network towards attacks by malicious IP addresses.

1.2 Packet Capture

Packet capture is the act of capturing data packets crossing a network. Deep packet capture (DPC) is the act of capturing complete network packets (header and payload) crossing a network. Once captured and stored, either in short-term memory or long-term storage, software tools can perform Deep packet inspection (DPI) to review network packet data, perform forensics analysis to uncover the root cause of network problems, identify security threats, and ensure data communications and network usage complies with outlined policy. Some DPCs can be coupled with DPI and can as a result manage, inspect, and analyze all network traffic in real-time at wire speeds while keeping a historical archive of all network traffic for further analysis. Partial packet capture can record headers without recording the total content of datagrams. This can reduce storage requirements, and avoid legal problems, but yet have enough data to reveal the essential information required for problem diagnosis.

1.2.1 Filtering

Packet capture can either capture the entire data stream or capture a filtered portion.

- ***Complete capture:***

  Packet capture has the ability to capture packet data from the data link layer on up (layers 2-7) of the OSI model. This includes headers and payload. Headers include information about what is contained in the packet and could be synonymous to an address or other printed information on the outside of an envelope. The payload includes the actual content of the packet and therefore synonymous to the contents of the envelope. Complete capture encompasses every packet that crosses a network segment, regardless of source, protocol or other distinguishing bits of data in the packet. Complete capture is the unrestricted, unfiltered, raw capture of all network packets.

- ***Filtered capture:***

  DPC devices may have the ability to limit capture of packets by protocol, IP address, MAC address, etc. With the application of filters, only complete packets that meet the criteria of the filter (header and payload) are captured, diverted, or stored.

1.2.2 Historical capture and analysis

Once data is captured, it can be analyzed right away or stored and analyzed later.

Many deep packet inspection tools rely on real-time inspection of data as it crosses the network, using known criteria for analysis. DPI tools make real-time decisions on what to do with packet data, perform designated analysis and act on the results. If packets are not stored after capture, they may be flushed away and actual packet contents are no longer available. Short-term capture and analysis tools can typically detect threats only when the triggers are known in advance but can act in real-time.

Historical capture and analysis stores all captured packets for further analysis, after the data has already crossed the network. As DPI and analysis tools deliver alerts, the historical record can be analyzed to apply context to the alert.

1.2.3 Identifying security breaches

Analysis of historical data captured with DPC assists in pinpointing the source of the intrusion. DPC can capture network traffic accessing certain servers and other systems to verify that the traffic flows belong to authorized employees. However this technique cannot function as an intrusion prevention system.

CHAPTER 2 : FEASIBILITY STUDY

Software Feasibility has four solid dimensions:

- ***TECHNOLOGY*****:** Is a project technically feasible? Is it within the state of art? Can defects be reduced to level matching the application needs?
- ***FINANCE*****:** Is it financially feasible? Can development be completed at a cost the software organization, it's client or the market can afford?
- ***TIME*****:** Will the project time to market beat the competition?
- ***RESOURCES*****:** Does the organization have the resources needed to succeed?

2.1 TECHNICAL FEASIBILITY

SOFWARE REQUIREMENTS**:**

- Operating System: Windows 7
- Development Platform: Netbeans IDE 7.3.1
- Development Language: JAVA
- Winpcap and IPSECCMD windows utilities have to be installed to enable java application to capture and block packets over the windows platform.
- "JPCAP" open source library for capturing and sending network packets from Java applications has to be installed.

HARDWARE REQUIREMENTS**:**

- Processor: AMD/Intel 2.4 GHz
- RAM: 2 GB
- Ethernet Card

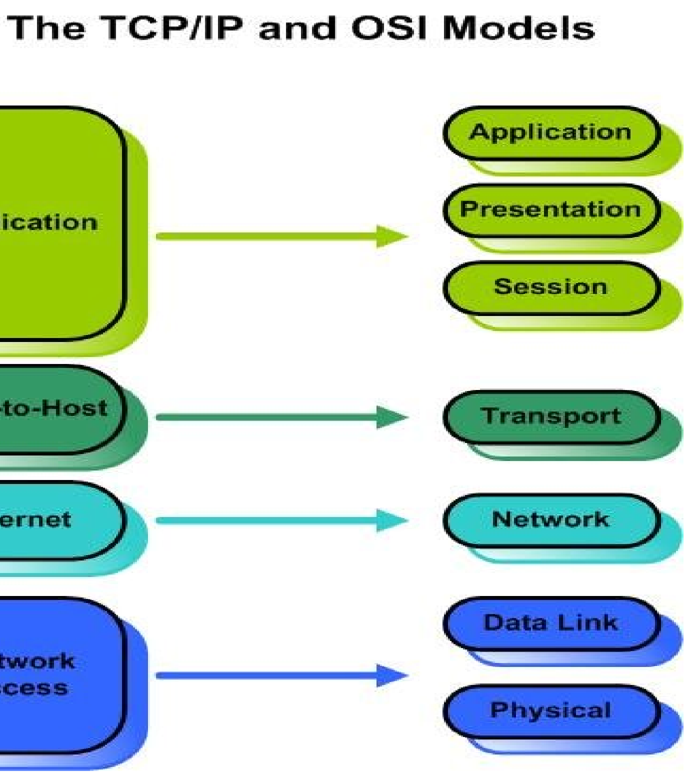
2.2 COST FEASIBILITY

The costing feasibility of the project can be estimated using current estimation models such as Lines of Code, which allow us to estimate cost as a function of size. Thus, this also allows us to estimate and analyze the feasibility of competition of the system in the given timeframe. This allows us to have a realistic estimate as well as a continuous evaluative perspective of the progress of the project.

Chapter 3 : ANALYSIS

3.1 TCP/IP Model

The TCP/IP model is a description framework for computer network protocols created in the 1970s by DARPA, an agency of the United States Department of Defense. It evolved from ARPANET, which was the world's first wide area network and a predecessor of the Internet. The TCP/IP Model is sometimes called the Internet Reference Model or the DoD Model. The TCP/IP model, or Internet Protocol Suite, describes a set of general design guidelines and implementations of specific networking protocols to enable computers to communicate over a network. TCP/IP provides end-to-end connectivity specifying how data should be formatted, addressed, transmitted, routed and received at the destination. Protocols exist for a variety of different types of communication services between computers.TCP/IP is generally described as having four abstraction layers . This layer architecture is often compared with the seven-layer OSI Reference Model formalized after the TCP/IP specifications. The TCP/IP model and related protocols are maintained by the IETF



3.1.1Key architectural principles

An early architectural document, emphasizes architectural principles over layering.

- End-to-End Principle: This principle has evolved over time. Its original expression put the maintenance of state and overall intelligence at the edges, and assumed the Internet that connected the edges retained no state and concentrated on speed and simplicity. Real-world needs for firewalls, network address translators, web content caches and the like have forced changes in this principle.
- Robustness Principle: "In general, an implementation must be conservative in its sending behavior, and liberal in its receiving behavior. That is, it must be careful to send well-formed datagrams, but must accept any datagram that it can interpret (e.g., not object to technical errors where the meaning is still clear) . "The second part of the principle is almost as important: software on other hosts may contain deficiencies that make it unwise to exploit legal but obscure protocol features."

As with all other communications protocol, TCP/IP is composed of layers:

- IP - is responsible for moving packet of data from node to node. IP forwards each packet based on a four byte destination address (the IP number). The Internet authorities assign ranges of numbers to different organizations. The organizations assign groups of their numbers to departments. IP operates on gateway machines that move data from department to organization to region and then around the world.
- TCP - is responsible for verifying the correct delivery of data from client to server. Data can be lost in the intermediate network. TCP adds support to detect errors or lost data and to trigger retransmission until the data is correctly and completely received.
- Sockets - is a name given to the package of subroutines that provide access to TCP/IP on most systems.

RFC 1122 on Host Requirements is structured in paragraphs referring to layers, but refers to many other architectural principles not emphasizing layering. It loosely defines a four-layer model, with the layers having names, not numbers, as follows:

- *Application (process-to-process) Layer*: This is the scope within which applications create user data and communicate this data to other processes or applications on another or the same host. The communications partners are often called peers. This is where the "higher level" protocols such as SMTP, FTP, SSH, HTTP, etc. operate.
- *Transport (host-to-host) Layer*: The Transport Layer constitutes the networking regime between two network hosts, either on the local network or on remote networks

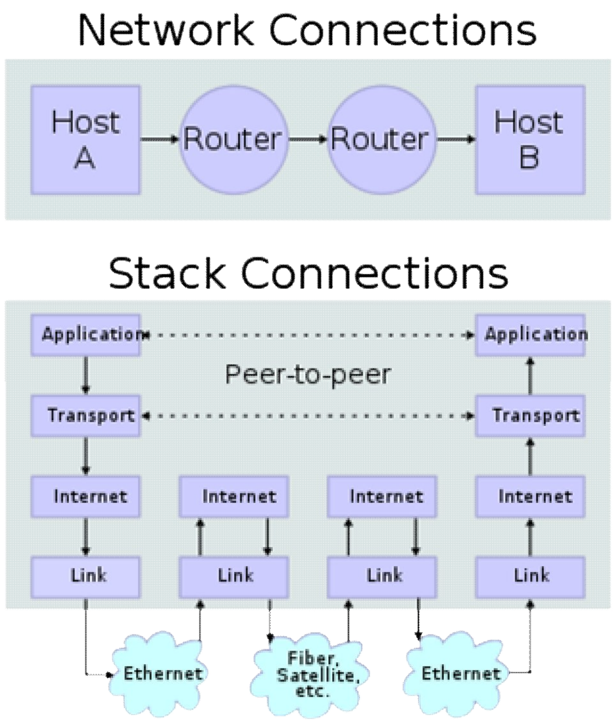
separated by routers. The Transport Layer provides a uniform networking interface that hides the actual topology (layout) of the underlying network connections. This is where flow-control, error-correction, and connection protocols exist, such as TCP. This layer deals with opening and maintaining connections between Internet hosts.

- *Internet (internetworking) Layer*: The Internet Layer has the task of exchanging datagrams across network boundaries. It is therefore also referred to as the layer that establishes internetworking, indeed, it defines and establishes the Internet. This layer defines the addressing and routing structures used for the TCP/IP protocol suite. The primary protocol in this scope is the Internet Protocol, which defines IP addresses. Its function in routing is to transport datagrams to the next IP router that has the connectivity to a network closer to the final data destination.
- *Link Layer:* This layer defines the networking methods with the scope of the local network link on which hosts communicate without intervening routers. This layer describes the protocols used to describe the local network topology and the interfaces needed to affect transmission of Internet Layer datagrams to next-neighbor hosts. (cf. the OSI Data Link Layer).

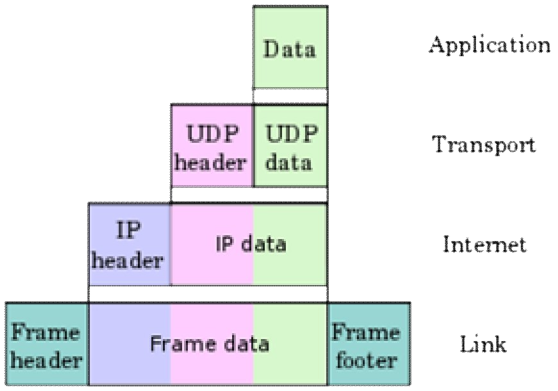
The Internet Protocol Suite and the layered protocol stack design were in use before the OSI model was established. Since then, the TCP/IP model has been compared with the OSI model in books and classrooms, which often results in confusion because the two models use different assumptions, including about the relative importance of strict layering.

3.1.2 Layers in the TCP/IP model

The layers near the top are logically closer to the user application, while those near the bottom are logically closer to the physical transmission of the data. Viewing layers as providing or consuming a service is a method of abstraction to isolate upper layer protocols from the nitty-gritty detail of transmitting bits over, for example, Ethernet and collision



Two Internet hosts connected via two routers and the corresponding layers used at each hop.



Encapsulation of application data descending through the TCP/IP layers

detection, while the lower layers avoid having to know the details of each and every application and its protocol.

This abstraction also allows upper layers to provide services that the lower layers cannot, or choose not to, provide. Again, the original OSI Reference Model was extended to include connectionless services. For example, IP is not designed to be reliable and is a best effort delivery protocol. This means that all transport layer implementations must choose whether or not to provide reliability and to what degree. UDP provides data integrity (via a checksum) but does not guarantee delivery; TCP provides both data integrity and delivery guarantee (by retransmitting until the receiver acknowledges the reception of the packet). This model lacks the formalism of the OSI reference model and associated documents, but the IETF does not use a formal model and does not consider this a limitation, as in the comment by David D. Clark, "We reject: kings, presidents and voting. We believe in: rough consensus and running code." Criticisms of this model, which have been made with respect to the OSI Reference Model, often do not consider ISO's later extensions to that model.

For multi-access links with their own addressing systems (e.g. Ethernet) an address mapping protocol is needed. Such protocols can be considered to be below IP but above the existing link system. While the IETF does not use the terminology, this is a subnetwork dependent convergence facility according to an extension to the OSI model, the Internal Organization of the Network Layer. ICMP & IGMP operate on top of IP but do not transport data like UDP or TCP. Again, this functionality exists as layer management extensions to the OSIRM MF. The SSL/TLS library operates above the transport layer (utilizes TCP) but below application protocols. The IETF explicitly does not intend to discuss transmission systems, which is a less academic but practical alternative to the OSI Reference Model.

3.1.3 Hardware and software implementation

Normally, application programmers are concerned only with interfaces in the Application Layer and often also in the Transport Layer, while the layers below are services provided by the TCP/IP stack in the operating system. Microcontroller firmware in the network adapter typically handles link issues, supported by driver software in the operational system. Non-programmable analog and digital electronics are normally in charge of the physical components in the Link Layer, typically using an application-specific integrated circuit (ASIC) chipset for each network interface or other physical standard.

However, hardware or software implementation is not stated in the protocols or the layered reference model. High-performance routers are to a large extent based on fast non-programmable digital electronics, carrying out link level switching.

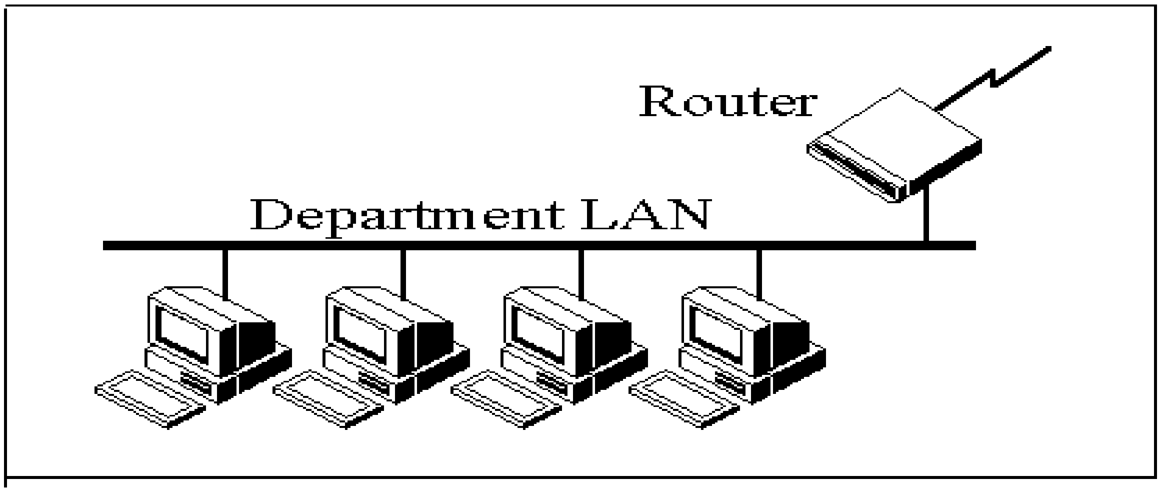
3.1.4 OSI and TCP/IP layering differences

The three top layers in the OSI model—the Application Layer, the Presentation Layer and the Session Layer—are not distinguished separately in the TCP/IP model where it is just the Application Layer. While some pure OSI protocol applications, such as X.400, also combined them, there is no requirement that a TCP/IP protocol stack needs to impose monolithic architecture above the Transport Layer.

The Session Layer roughly corresponds to the Telnet virtual terminal functionality, which is part of text based protocols such as the HTTP and SMTP TCP/IP model Application Layer protocols. It also corresponds to TCP and UDP port numbering, which is considered as part of the transport layer in the TCP/IP model.

The presentation layer has similarities to the MIME standard, which also is used in HTTP and SMTP. Since the IETF protocol development effort is not concerned with strict layering, some of its protocols may not appear to fit cleanly into the OSI model. These conflicts, however, are more frequent when one only looks at the original OSI model, ISO 7498, without looking at the annexes to this model (e.g., ISO 7498/4 Management Framework), or the ISO IONL. When the IONL and Management Framework documents are considered, the ICMP and IGMP are neatly defined as layer management protocols for the network layer. In like manner, the IONL provides a structure for "subnetwork dependent convergence facilities" such as ARP and RARP.

IETF protocols can be encapsulated recursively, as demonstrated by tunneling protocols such as Generic Routing Encapsulation (GRE). While basic OSI documents do not consider tunneling, there is some concept of tunneling in yet another extension to the OSI architecture, specifically the transport layer gateways within the International Standardized Profile framework [11]. The associated OSI development effort, however, has been abandoned given the overwhelming adoption of TCP/IP protocols.



3.1.5 Addresses

Each technology has its own convention for transmitting messages between two machines within the same network. On a LAN, messages are sent between machines by supplying the six byte unique identifier (the "MAC" address). In an SNA network, every machine has Logical Units with their own network address. DECNET, Appletalk, and Novell IPX all have a scheme for assigning numbers to each local network and to each workstation attached to the network. On top of these local or vendor specific network addresses, TCP/IP assigns a unique number to every workstation in the world. This "IP number" is a four byte value that, by convention, is expressed by converting each byte into a decimal number (0 to 255) and separating the bytes with a period. For example, the PC Lube and Tune server is 130.132.59.234.

3.1.6 Subnets

Although the individual subscribers do not need to tabulate network numbers or provide explicit routing, it is convenient for most Class B networks to be internally managed as a much smaller and simpler version of the larger network organizations. It is common to subdivide the two bytes available for internal assignment into a one byte department number and a one byte workstation ID. The enterprise network is built using commercially available TCP/IP router boxes. Each router has small tables with 255 entries to translate the one byte department number into selection of a destination Ethernet connected to one of the routers. Mssages to the PC Lube and Tune server (130.132.59.234) are sent through the national and New England regional networks based on the 130.132 part of the number.

Chapter 4: JAVA CLASS LIBRARY: JPCAP

Introduction

Jpcap is a Java library for capturing and sending network packets.

Using Jpcap, you can develop applications to capture packets from a network interface and visualize/analyze them in Java. You can also develop Java applications to send arbitrary packets through a network interface. Jpcap has been tested on Microsoft Windows (98/2000/XP/Vista/7/-), Linux (Fedora, Mandriva, Ubuntu), Mac OS X (Darwin), FreeBSD, and Solaris. Jpcap can capture Ethernet, IPv4, IPv6, ARP/RARP, TCP, UDP, and ICMPv4 packets. Jpcap is open source, and is licensed under GNU LGPL. Jpcap is based on libpcap/winpcap, and is implemented in C and Java.

Functionalities provided by JPCAP include :

- 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 can be used to develop many kinds of network applications, including :

- network and protocol analyzers
- network monitors
- traffic loggers
- traffic generators
- user-level bridges and routers
- network intrusion detection systems (NIDS)
- network scanners
- security tools

4.1 Basic functions provided by Jpcap

4.1.1 Obtain the list of network interfaces

When you want to capture packets from a network, the first thing you have to do is to obtain the list of network interfaces on your machine. To do so, Jpcap provides JpcapCaptor.getDeviceList() method. It returns an array of Network Interface objects. A NetworkInterface object contains some information about the corresponding network interface, such as its name, description, IP and MAC addresses, and datalink name and description.

public static NetworkInterface[] **getDeviceList**()

Returns the interfaces that can be used for capturing.

Returns:

List of Interface objects

4.1.2 Open a network interface

Once you obtain the list of network interfaces and choose which network interface to capture packets from, you can open the interface by using JpcapCaptor.openDevice() method.

```
NetworkInterface[] devices = JpcapCaptor.getDeviceList();
int index=...;  // set index of the interface that you want to open.

//Open an interface with openDevice(NetworkInterface intrface, int snaplen, boolean
promics, int to_ms)
JpcapCaptor captor=JpcapCaptor.openDevice(device[index], 65535, false, 20);
```

4.1.3 Capture packets from the network interface

Once you obtain an instance of of JpcapCaptor, you can capture packets from the interface.

There are two major approaches to capture packets using a JpcapCaptor instance: using a callback method, and capturing packets one-by-one.

Using a callback method

In this approach, you implement a callback method to process captured packets, and then pass the callback method to Jpcap so that Jpcap calls it back every time it captures a packet. Let's see how you can do this approach in detail.

First, you implement a callback method by defining a new class which implements the PacketReceiver interface. The PacketReceiver interface defines a receivePacket() method, so you need to implement a receivePacket() method in your class.

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

Capturing packets one-by-one

Using a callback method is a little bit tricky because you don't know when the callback method is called by Jpcap. If you don't want to use a callback method, you can also capture packets using the JpcapCaptor.getPacket() method.

getPacket() method simply returns a captured packet. You can (or have to) call getPacket() method multiple times to capture consecutive packets.

```
JpcapCaptor captor=JpcapCaptor.openDevice(device[index], 65535, false, 20);

for(int i=0;i<10;i++){
  //capture a single packet and print it out
  System.out.println(captor.getPacket());
}

captor.close();
```

4.1.4 Set capturing filter

In Jpcap, you can set a filter so that Jpcap doesn't capture unwanted packets. For example, if you only want to capture TCP/IPv4 packets, you can set a filter as following:

```
JpcapCaptor captor=JpcapCaptor.openDevice(device[index], 65535, false, 20);
//set a filter to only capture TCP/IPv4 packets
captor.setFilter("ip and tcp", true);
```

The filter expression "ip and tcp" means to "keep only the packets that are both IPv4 and TCP and deliver them to the application". By properly setting a filter, you can reduce the number of packets to examine, and thus can improve the performance of your application.

4.1.5 Save captured packets into a file

You can save captured packets into a binary file so that you can later retrieve them using Jpcap or other applications which supports reading a tcpdump format file.

To save captured packets, you first need to open a file by calling JpcapWriter.openDumpFile() method with an instance of JpcapCaptor which was used to capture packets, and a String filename.

```
JpcapCaptor captor=JpcapCaptor.openDevice(device[index], 65535, false, 20);
//open a file to save captured packets
JpcapWriter writer=JpcapWriter.openDumpFile(captor,"yourfilename");
```

Once you obtained an instance of JpcapWriter through openDumpFile() method, you can save captured packets using JpcapWriter.writePacket() method. After you saved all the packets you want to save, you need to call JpcapWriter.close() method to close the opened file.

Chapter 5: WINDOWS UTILITY

5.1 WINPCAP

WinPcap is an open source library for packet capture and network analysis for the Win32 platforms.

Most networking applications access the network through widely used operating system primitives such as sockets. It is easy to access data on the network with this approach since the operating system copes with the low level details (protocol handling, packet reassembly, etc.) and provides a familiar interface that is similar to the one used to read and write files.

Sometimes, however, the 'easy way' is not up to the task, since some applications require direct access to packets on the network. That is, they need access to the "raw" data on the network without the interposition of protocol processing by the operating system.

The purpose of WinPcap is to give this kind of access to Win32 applications; it provides facilities to:

- capture raw packets, both the ones destined to the machine where it's running and the ones exchanged by other hosts (on shared media)
- filter the packets according to user-specified rules before dispatching them to the application
- transmit raw packets to the network
- gather statistical information on the network traffic

5.1.1 What kind of programs use WinPcap:

The WinPcap programming interface can be used by many types of network tools for analysis, troubleshooting, security and monitoring. In particular, classical tools that rely on WinPcap are:

- network and protocol analyzers
- network monitors
- traffic loggers
- traffic generators
- user-level bridges and routers

- network intrusion detection systems (NIDS)
- network scanners
- security tools

5.1.2 What WinPcap can't do:

WinPcap receives and sends the packets *independently* from the host protocols, like TCP-IP. This means that it isn't able to 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.

5.1.3 WinPcap consists of:

- Drivers for Windows 95/98/Me , and for the Windows NT family (Windows NT 4.0, Windows 2000, Windows XP, Windows Server 2003, Windows7, etc.), which use NDIS to read packets directly from a network adapter;
- Implementations of a lower-level library for the listed operating systems, to communicate with those drivers;
- A port of libpcap that uses the API offered by the low-level library implementations.

5.2 IPSEC(Internet Protocol Security)

Internet Protocol security (IPSec) filtering rules can be used to help protect Windows 2000-based, Windows XP-based, and Windows 7-based computers from network-based attacks from threats such as viruses and worms. It filter a particular protocol and port combination for both inbound and outbound network traffic.

IPSec policies can be applied locally or be applied to a member of a domain as part of that domain's group policies. Local IPSec policies can be *static* (persistent after restarts) or *dynamic* (volatile). Static IPSec policies are written to the local registry and persist after the operating system is restarted. Dynamic IPSec policies are not permanently written to the registry and are removed if the operating system or the IPSec Policy Agent service is restarted.

IPSec filter rules can cause network programs to lose data and to stop responding to network requests, including failure to authenticate users. Use IPSec filter rules as a defensive measure of last resort and only after you have a clear understanding of the impact that blocking specific ports will have in your environment.

5.2.1 Determine whether an IPSec policy is assigned

5.2.1.1 Windows based computers

Before you create or assign any new IPSec policies to a Windows based computer, determine whether any IPSec policies are being applied from the local registry or through a GPO. To do this, follow these steps:

1. Install Netdiag.exe from the Windows XP CD by running Setup.exe from the Support\Tools folder.
2. Open a command prompt, and then set the working folder to C:\Program Files\Support Tools.
3. Run the following command to verify that there is not an existing IPSec policy already assigned to the computer:

   netdiag/test:ipsec

   If no policy is assigned, you receive the following message:

   IP Security test . . . . : Passed IPSec policy service is active, but no policy is assigned.

5.2.2 Create a static policy to block traffic

5.2.2.1Windows Server 2003-based and Windows XP/Vista/7-based computers

For systems that do not have a locally defined IPSec policy enabled, create a new local static policy to block traffic that is directed to a specific protocol and a specific port on Windows based computers. To do this, follow these steps:

1. Verify that the IPSec Policy Agent service is enabled and started in the Services MMC snap-in.

2. Install IPSeccmd.exe. IPSeccmd.exe is part of Windows XP Service Pack 2 (SP2) Support Tools.

   Note IPSeccmd.exe will run on Windows XP/7 and Windows Server 2003 operating systems, but the tool is only available from the Windows XP SP2 Support Tools package.

3. Open a command prompt, and then set the working folder to the folder where you installed the Windows XP Service Pack 2 Support Tools.

   Note The default folder for Windows XP SP2 Support Tools is C:\Program Files\Support Tools.

4. To create a new local IPSec policy and filtering rule that applies to network traffic from any IP address to the IP address of the Windows Server 2003-based or Windows XP/7 -based computer that you are configuring, use the following command.

   Note In the following command, *Protocol* and *PortNumber* are variables.

   IPSeccmd.exe -w REG -p "Block *ProtocolPortNumber* Filter" -r "Block Inbound *ProtocolPortNumber* Rule" -f *=0:*PortNumber*:*Protocol* -n BLOCK –x

   For example, to block network traffic from any IP address and any source port to destination port UDP 1434 on a Windows Server 2003-based or Windows 7-based computer, type the following.

   IPSeccmd.exe -w REG -p "Block UDP 1434 Filter" -r "Block Inbound UDP 1434 Rule" -f *=0:1434:UDP -n BLOCK -x

   The following example blocks inbound access to TCP port 80 but still allows outbound TCP 80 access. This policy is sufficient to help protect computers that run Microsoft Internet Information Services (IIS) 5.0 from the "Code Red" worm and the "Nimda" worm.

   IPSeccmd.exe -w REG -p "Block TCP 80 Filter" -r "Block Inbound TCP 80 Rule" -f *=0:80:TCP -n BLOCK -x

Note The -x switch assigns the policy immediately. If you enter this command, the "Block UDP 1434 Filter" policy is unassigned and the "Block TCP 80 Filter" is assigned. To add the policy but not assign the policy, type the command without the -x switch at the end.

5. To add an additional filtering rule to the existing "Block UDP 1434 Filter" policy that blocks network traffic that originates from your Windows Server 2003-based or Windows XP/7-based computer to any IP address, use the following command.

   Note In this command, *Protocol* and *PortNumber* are variables:

   IPSeccmd.exe -w REG -p "Block *ProtocolPortNumber* Filter" -r "Block Outbound *ProtocolPortNumber* Rule" -f *0=:*PortNumber*:*Protocol* -n BLOCK

   For example, to block any network traffic that originates from your Windows Server 2003-based or Windows XP-based computer that is directed to UDP 1434 on any other host, type the following. This policy is sufficient to help prevent computers that run SQL Server 2000 from spreading the "Slammer" worm.

   IPSeccmd.exe -w REG -p "Block UDP 1434 Filter" -r "Block Outbound UDP 1434 Rule" -f 0=*:1434:UDP -n BLOCK

   Note You can add as many filtering rules to a policy as you want by using this command. For example, you can use this command to block multiple ports by using the same policy.

6. The policy in step 5 will now be in effect and will persist every time that the computer is restarted. However, if a domain-based IPSec policy is assigned to the computer later, this local policy will be overridden and will no longer apply.

   To verify the successful assignment of your filtering rule, set the working folder to C:\Program Files\Support Tools at the command prompt, and then type the following command:

   netdiag /test:ipsec /debug

If policies for both inbound and outbound traffic are assigned as in these examples, you will receive the following message:

```
IP Security test . . . . . . . . . :
Passed Local IPSec Policy Active: 'Block UDP 1434 Filter' IP Security Policy Path:
SOFTWARE\Policies\Microsoft\Windows\IPSec\Policy\Local\ipsecPolicy{D239C59
9-F945-47A3-A4E3-B37BC12826B9}

There are 2 filters
No Name
Filter Id: {5EC1FD53-EA98-4C1B-A99F-6D2A0FF94592}
Policy Id: {509492EA-1214-4F50-BF43-9CAC2B538518}
Src Addr : 0.0.0.0 Src Mask : 0.0.0.0
Dest Addr : 192.168.1.1 Dest Mask : 255.255.255.255
Tunnel Addr : 0.0.0.0 Src Port : 0 Dest Port : 1434
Protocol : 17 TunnelFilter: No
Flags : Inbound Block
No Name
Filter Id: {9B4144A6-774F-4AE5-B23A-51331E67BAB2}
Policy Id: {2DEB01BD-9830-4067-B58A-AADFC8659BE5}
Src Addr : 192.168.1.1 Src Mask : 255.255.255.255
Dest Addr : 0.0.0.0 Dest Mask : 0.0.0.0
Tunnel Addr : 0.0.0.0 Src Port : 0 Dest Port : 1434
Protocol : 17 TunnelFilter: No
Flags : Outbound Block
```

Note The IP addresses and graphical user interface (GUID) numbers will be different based on your Windows Server 2003-based or Windows XP/7-based computer.

5.2.3 Add a block rule for a specific protocol and port :

5.2.3.1 Windows Server 2003-based and Windows XP/7-based computers

To add a block rule for a specific protocol and port on a Windows Server 2003-based or Windows XP/7-based computer that has an existing locally-assigned static IPSec policy, follow these steps:

1. Install IPSeccmd.exe. IPSeccmd.exe is part of Windows XP SP2 Support Tools.

2. Identify the name of the currently assigned IPSec policy. To do this, type the following at a command prompt:

   ```
   netdiag /test:ipsec
   ```

   If a policy is assigned, you will receive a message that is similar to the following:

   ```
   IP Security test . . . . . . . . . : Passed
   Local IPSec Policy Active: 'Block UDP 1434 Filter'
   ```

3. If there is an IPSec policy already assigned to the computer (local or domain), use the following command to add an additional BLOCK Filter Rule to the existing IPSec policy.

   Note In this command, *Existing_IPSec_Policy_Name*, *Protocol*, and *PortNumber* are variables.

   IPSeccmd.exe -p "*Existing_IPSec_Policy_Name*" -w REG -r "Block *ProtocolPortNumber* Rule" -f *=0:*PortNumber*:*Protocol* -n BLOCK

   For example, to add a Filter Rule to block inbound access to TCP port 80 to the existing Block UDP 1434 Filter, type the following command:

   ```
   IPSeccmd.exe -p "Block UDP 1434 Filter" -w REG -r "Block Inbound TCP 80 Rule" -f *=0:80:TCP -n BLOCK
   ```

5.2.4 Add a dynamic block policy for a specific protocol and port:

5.2.4.1 Windows Server 2003 and Windows XP-based computers

You may want to temporarily block access to a specific port. For example, you may want to block a specific port until you can install a hotfix or if a domain-based IPSec policy is already assigned to the computer. To temporarily block access to a port on computers by using IPSec policy, follow these steps:

1. Install IPSeccmd.exe. IPSeccmd.exe is part of Windows XP Service Pack 2 Support Tools.

   Note IPSeccmd.exe will run on Windows operating systems, but the tool is only available from the Windows XP SP2 Support Tools package.

2. To add a dynamic BLOCK filter that blocks all packets from any IP address to your system's IP address and targeted port, type the following at a command prompt.

   Note In the following command, *Protocol* and *PortNumber* are variables.

   IPSeccmd.exe -f [*=0:*PortNumber*:*Protocol*]

   Note This command creates the block filter dynamically. The policy remains assigned as long as the IPSec Policy Agent service is running. If the IPSec Policy Agent service is restarted or the computer is restarted, this policy is lost. If you want to dynamically reassign the IPSec Filtering Rule every time that the system is restarted, create a startup script to reapply the Filter Rule. If you want to permanently apply this filter, configure the filter as a static IPSec policy. The IPSec Policy Management MMC snap-in provides a graphical user interface for managing IPSec policy configuration. If a domain-based IPSec policy is already applied, the netdiag /test:ipsec /debug command may only show the filter details if the command is executed by a user who has domain administrator credentials.

5.2.5 IPSec filtering rules and Group Policy

For environments where IPSec policies are assigned by a Group Policy setting, you have to update the whole domain's policy to block the particular protocol and port. After you successfully configure the Group Policy IPSec settings, you must enforce a refresh of the Group Policy settings on all the Windows Server 2003-based, Windows XP/vista/7-based, and Windows 2000-based computers in the domain. To do this, use the following command:
secedit /refreshpolicy machine_policy

The IPSec policy change will be detected within one of two different polling intervals. For a newly assigned IPSec policy being applied to a GPO, the IPSec policy will be applied to the clients within the time set for the Group Policy polling interval or when the secedit /refreshpolicy machine_policy command is run on the client computers. If IPSec policy is already assigned to a GPO and new IPSec filters or rules are being added to an existing policy, the secedit command will not make IPSec recognize changes. In this scenario, modifications to an existing GPO-based IPSec policy will be detected within that IPSec policy's own polling interval. This interval is specified on the General tab for that IPSec policy. You can also force a refresh of the IPSec Policy settings by restarting the IPSec Policy Agent service. If the IPSec service is stopped or restarted, IPSec-secured communications will be interrupted and will take several seconds to resume. This may cause program connections to disconnect, particularly for connections that are actively transferring large volumes of data. In situations where the IPSec policy is applied only on the local computer, you do not have to restart the service.

5.2.6 Unassign and delete an IPSec policy:

5.2.6.1 Windows Server 2003-based and Windows XP/vista/7-based computers

5.2.6.1.1 Computers that have a locally-defined static policy

Open a command prompt, and then set the working folder to the folder where you installed Ipsecpol.exe.

1. To unassign the filter that you created earlier, use the following command:

   IPSeccmd.exe -w REG -p "Block *ProtocolPortNumber* Filter" –y

   For example, to unassign the Block UDP 1434 Filter that you created earlier, use the following command:

   IPSeccmd.exe -w REG -p "Block UDP 1434 Filter" -y

2. To delete the filter that you created, use the following command:

   IPSeccmd.exe -w REG -p "Block *ProtocolPortNumber* Filter" -r "Block *ProtocolPortNumber* Rule" –o

   For example, to delete the "Block UDP 1434 Filter" filter and both of the rules that you created, use the following command:

   IPSeccmd.exe -w REG -p "Block UDP 1434 Filter" -r "Block Inbound UDP 1434 Rule" -r "Block Outbound UDP 1434 Rule" –o

5.2.6.1.2 Computers that have a locally-defined dynamic policy

Dynamic IPSec policy is unapplied if the IPSec Policy Agent service is stopped by using the net stop policyagent command. To delete the specific commands that were used without stopping the IPSec Policy Agent service, following these steps:

3. Open a command prompt, and then set the working folder to the folder where you installed Windows XP Service Pack 2 Support Tools.
4. Type the following command:

   IPSeccmd.exe –u

   Note You can also restart the IPSec Policy Agent service to clear all dynamically-assigned policies.

5.2.7 APPLIES TO

- Microsoft Windows 2000 Professional Edition
- Microsoft Windows 2000 Server
- Microsoft Windows 2000 Advanced Server
- Microsoft Windows 2000 Datacenter Server
- Microsoft Windows Server 2003, Datacenter Edition (32-bit x86)
- Microsoft Windows Server 2003, Enterprise Edition (32-bit x86)
- Microsoft Windows Server 2003, Standard Edition (32-bit x86)
- Microsoft Windows Server 2003, Web Edition
- Microsoft Windows XP Home Edition
- Microsoft Windows XP Professional
- Microsoft Windows 7

Chapter 6: PACKET

Each logical network uses discrete data messages called packets, as defined in the last chapter. The packets can be actual messages on the transmission line (which has a lot more information included) or simply the message you're sending.

The logical network packet at the generic level consists of information about the source, destination, and data payload. Each logical network offers varying degrees of features and interfaces (protocols). All packet types and protocols are available with network programming. Each type has significant strengths and weaknesses. Like shopping for tools, your choice of packet type depends on how you use it.

You can choose from four basic Internet packet protocols: raw IP, ICMP, UDP (unreliable messaging), and TCP (streaming) all layered on top of the physical network

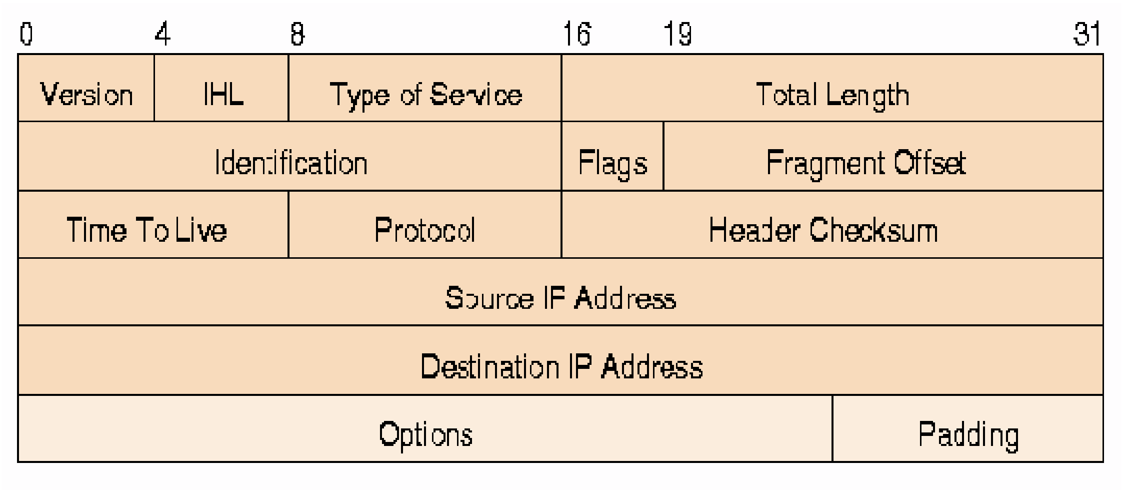
Each protocol is very different, but they all share one common necessary feature: They all carry the program's message. Some protocols include a source address, while some require a destination. You may think that not requiring a destination is a little unusual, but some protocols (like UUCP) use the connection as the address to the destination.

The Internet Protocol (IP) [RFC791] requires a packet to have three basic elements: source, destination, and data. (The data payload includes its length.) These elements provide the packet a level of autonomy. No matter where a packet is, you can identify where it came from, where it's going, and how big it is.

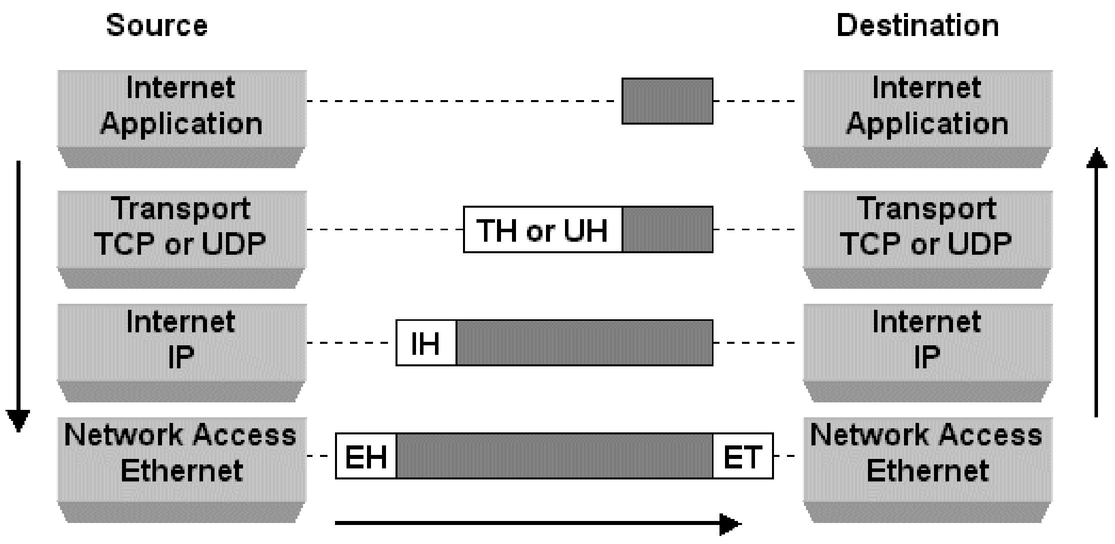
The packet's autonomy is an important feature of the Internet. As long as the packet is *alive* (the data is timely and relevant), routers can move data to its destination when the packet is launched onto the network.

6.1 IP PACKET STRUCTURE

All IP packets are structured the same way - an IP header followed by a variable-length data field. A summary of the contents of the internet header follows:



Each IP (Internet Protocol) packet consists of a header followed by a data field. The header length can vary between 20 and 60 bytes, and the total size of the packet can be up to 65535 bytes. However, many systems cannot handle packets as large as the protocol allows and a working maximum size is 576 bytes. The header must have 5 words (of 32 bits each) of defined contents, and may have up to 10 more words of optional information. Most network data transmission technologies use packets to transmit data from a source device to destination. The IP protocol is not exception. IP packets are the most important and fundamental components of the protocol. The two main functions of the IP protocol are routing and addressing. To route packets to and from machines on a network, IP uses IP addresses which are carried along in the packets. A lot of other information is carried along as well, in the packet header. The structure of an IP packet is shown in the picture here.



The identification tag is used to help reassemble the packet from several eventual fragments.
The flag states whether the packet can be fragmented or not.
The fragment offset is a field to identify which fragment this packet is attached to.
Time to Live (TTL) is a number that indicates how many hops the packet can make before it dies. This is done to prevent a packet from remaining forever on a network, thus causing congestion. TTL is decremented at each hop.
The header checksum is a number used for error detection and correction during packet transmission.

6.2 FLAGS

0	**1**	**2**
0	DF	MF

The first flag bit is reserved, and must be zero.
Second flag bit (DF): 0 = May Fragment, 1 = Don't Fragment..
Third flag bit (MF) 0 = Last Fragment, 1 = More Fragments.

There are six ‘control bits’ defined in TCP, one or more of which is defined in each packet. The control bits are ‘SYN’, ‘ACK’, ‘PSH’, ‘URG’, ‘RST’, and ‘FIN’. TCP uses these bits to define the purpose and contents of a packet.

SYN bit is used in establishing a TCP connection to synchronize the sequence numbers between both endpoints.

ACK bit is used to acknowledge the remote host's sequence numbers, declaring that the information in the acknowledgment field is valid.

PSH flag is set on the sending side, and tells the TCP stack to flush all buffers and send any outstanding data up to and including the data that had the PSH flag set. When the receiving TCP sees the PSH flag, it too must flush its buffers and pass the information up to the application.

URG bit indicates that the urgent pointer field has a valid pointer to data that should be treated urgently and be transmitted before non- urgent data.

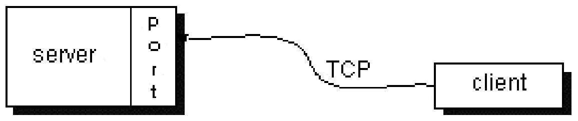
Reset or RST is used to reset the connection. If a station involved in a TCP session notices that it is not receiving acknowledgements for anything it sends, the connection is now unsynchronized, and the station should send a reset. This is a half-open connection where only one side is involved in the TCP session. This cannot work by definition of the protocol.

FIN bit is used to indicate that the client will send no more data (but will continue to listen for data).

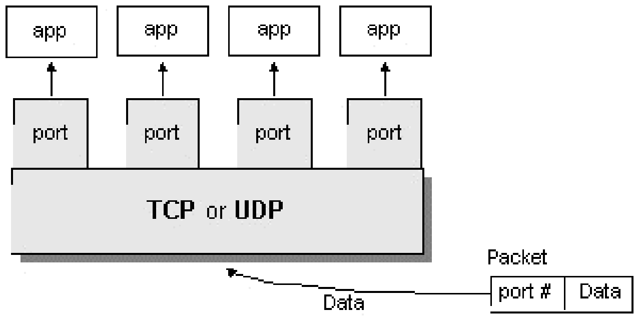
In the particular case of receiving one-way traffic, the RST flag should be used as it closes the connection.

6.3 PORTS

Generally speaking, a computer has a single physical connection to the network. All data destined for a particular computer arrives through that connection. However, the data may be intended for different applications running on the computer. So how does the computer know to which application to forward the data? Through the use of ports. Data transmitted over the Internet is accompanied by addressing information that identifies the computer and the port for which it is destined. The computer is identified by its 32-bit IP address, which IP uses to deliver data to the right computer on the network. Ports are identified by a 16-bit number, which TCP and UDP use to deliver the data to the right application. In connection-based communication such as TCP, a server application binds a socket to a specific port number. This has the effect of registering the server with the system to receive all data destined for that port. A client can then rendezvous with the server at the server's port, as illustrated here:



The TCP and UDP protocols use ports to map incoming data to a particular process running on a computer. In datagram-based communication such as UDP, the datagram packet contains the port number of its destination and UDP routes the packet to the appropriate application, as illustrated in this figure:



Port numbers range from 0 to 65,535 because ports are represented by 16-bit numbers. The port numbers ranging from 0 - 1023 are restricted; they are reserved for use by well-known services such as HTTP and FTP and other system services. These ports are called *well-known ports*. Your applications should not attempt to bind to them.

Chapter 7: PROTOCOL

In computing, a protocol is a convention or standard that controls or enables the connection, communication, and data transfer between computing endpoints. In its simplest form, a protocol can be defined as the rules governing the syntax, semantics, and synchronization of communication. Protocols may be implemented by hardware, software, or a combination of the two. At the lowest level, a protocol defines the behavior of a hardware connection. Typical properties. While protocols can vary greatly in purpose and sophistication, most specify one or more of the following properties:

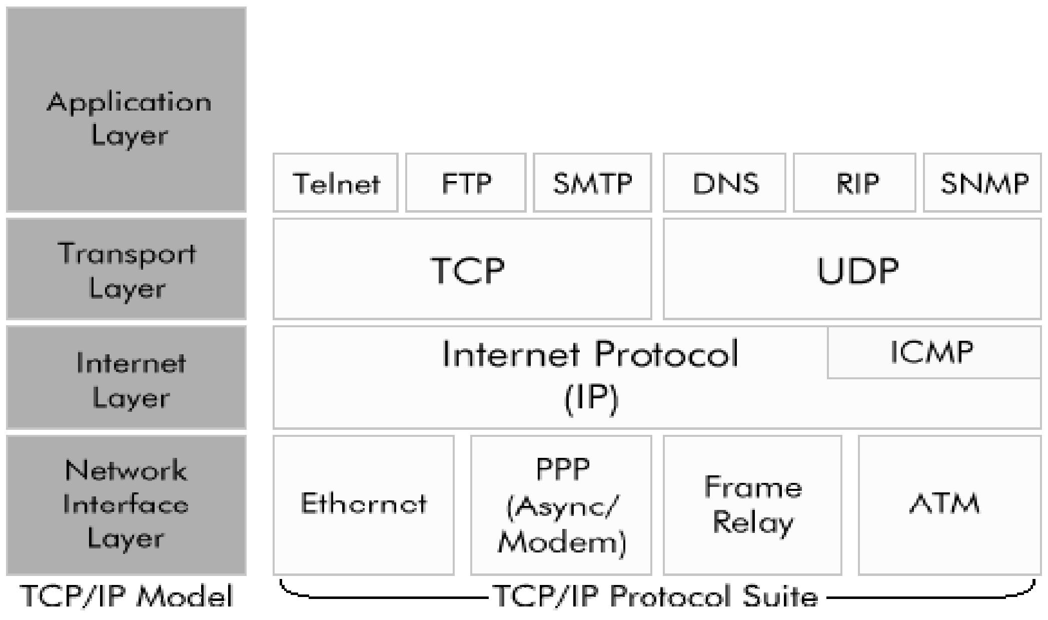
Detection of the underlying physical connection (wired or wireless), or the existence of the other endpoint or node

- Handshaking
- Negotiation of various connection characteristics
- How to start and end a message
- How to format a message
- What to do with corrupted or improperly formatted messages (error correction)
- How to detect unexpected loss of the connection, and what to do next
- Termination of the session and or connection.

7.1 Importance

The widespread use and expansion of communications protocols is both a prerequisite for the Internet, and a major contributor to its power and success. The pair of Internet Protocol (or IP) and Transmission Control Protocol (or TCP) are the most important of these, and the term TCP/IP refers to a collection (or protocol suite) of its most used protocols. Most of the Internet's communication protocols are described in the RFC documents of the Internet Engineering Task Force (or IETF).

The protocols in human communication are separate rules about appearance, speaking, listening and understanding. All these rules, also called protocols of conversation, represent different layers of communication. They work together to help people successfully communicate. The need for protocols also applies to network devices. Computers have no



way of learning protocols, so network engineers have written rules for communication that must be strictly followed for successful host-to-host communication. These rules apply to different layers of sophistication such as which physical connections to use, how hosts listen, how to interrupt, how to say good-bye, and in short how to communicate, what language to use and many others. These rules, or protocols, that work together to ensure successful communication are grouped into what is known as a protocol suite.

Object-oriented programming has extended the use of the term to include the programming protocols available for connections and communication between objects.

Generally, only the simplest protocols are used alone. Most protocols, especially in the context of communications or networking, are layered together into protocol stacks where the various tasks listed above are divided among different protocols in the stack.

Whereas the protocol stack denotes a specific combination of protocols that work together, a reference model is a software architecture that lists each layer and the services each should offer. The classic seven-layer reference model is the OSI model, which is used for conceptualizing protocol stacks and peer entities. This reference model also provides an opportunity to teach more general software engineering concepts like hiding, modularity, and delegation of tasks. This model has endured in spite of the demise of many of its protocols (and protocol stacks) originally sanctioned by the ISO.

7.1.2 Common protocols

- IP (Internet Protocol)
- UDP (User Datagram Protocol)
- TCP (Transmission Control Protocol)
- HTTP (Hypertext Transfer Protocol)
- FTP (File Transfer Protocol)
- Telnet (Telnet Remote Protocol)
- SSH (Secure Shell Remote Protocol)
- POP3 (Post Office Protocol 3)
- SMTP (Simple Mail Transfer Protocol)
- PPP (Point-to-Point Protocol)

Chapter 8: SNIFFING

Packet analyzer captures network packets and provide view for full TCP conversations and UDP threads The Packet sniffing and analysis features of the packet analyzer are unique. Combining packet sniffing and analysis with network statistics gathering and presentation, it makes the job for a network administrator easier. There are three types of sniffing methods. Some methods work in non-switched networks while others work in switched networks. The sniffing methods are: IP-based sniffing, MAC-based sniffing and ARP-based sniffing.

IP BASED PACKET SNIFFING

Sniff IP information with ease. IP Sniffer provides IP sniffing utility to help you work with IP addresses. The suit includes Domain-to-IP Converter, Batch Ping, Tracert, Whois, Website Scanner and Connection Monitor as well as an IP-to-Country Converter into a single interface. With the powerful IP & Web tool you can:

1. Perform batch and continuous pings on multiple servers;
2. Obtaining all available information on a given IP address or domain name such as Organization or the ISP that owns the IP address, including the country, state, city, address, contact phone numbers and e-mails;
3. Lookup IP address for a single or list of domain names and vice versa;
4. Find out the country associated with a single or list of domains or IP addresses;
5. Trace IP addresses to their destination and investigate connection problems;
6. Determine name, date, last-modified,version and operation system of the remote web server;
7. Allow you to scan any given web site and produce a list of links (including htm cgi php asp jsp jpg gif mp3 mpeg exe zip rar swf and more file types) found in the site, using several criteria to filter the results;
8. Monitor all the TCP/IP connections from your computer to the internet automatically;
9. Get all of the information about the website currently open in the Internet Explorer.

MAC BASED PACKET SNIFFING

MAC based packet sniffing intercepts all network activity; both incoming and outgoing. This means we can to see people’s IM conversations, emails sent and received, websites visited, and even pictures they are viewing from web pages. Additionally, we can also hack their usernames and passwords. As you can see from the images above, the data captured may not be exactly intuitive, but its easy enough to see from the first image the person is looking at a Google image search results page in Safari. Looking at images tab (2nd image) you can see the images that were downloaded from the images.google.com page.

ARP BASED PACKET SNIFFING

This method works a little different. It doesn't put the network card into promiscuous mode. This isn't necessary because ARP packets will be sent to us. This happens because the ARP protocol is stateless. Because of this, sniffing can be done on a switched network.
To perform this kind of sniffing, you first have to poison the ARP cache1 of the two hosts that you want to sniff, identifying yourself as the other host in the connection. Once the ARP caches are poisoned, the two hosts start their connection, but instead of sending the traffic directly to the other host it gets sent to us. We then log the traffic and forward it to the real intended host on the other side of the connection. This is called a man-in-the-middle attack. See Diagram 1 for a general idea of the way it works.

PACKET ANALYZER

Packet analyzer is a program that captures all of the packets of data that pass through a given network interface, and recognizes and decodes the packets of interest. It is sometimes referred to as a network monitor. The captured network data can be browsed via a GUI.

Working

- A computer connected to the LAN has 2 addresses.
- The Data Link Layer uses an Ethernet header with the MAC address of the destination machine
- The Network Layer maps IP network addresses to the MAC address as required by the Data Link Protocol
- It initially looks up the MAC address of the destination machine in the Address Resolution Protocol cache
- If no entry is found for the MAC address, the ARP broadcasts a request packet to all machines on the network
- The machine with that address responds to the source machine with its MAC address
- This MAC address then gets added to the source machines ARP Cache.
- This MAC address is then used by the source machine in all its communications with the destination machine

Chapter 9- FEATURES

FLAGGED HOSTS

It is implemented for the purpose of security. The suspected hosts are marked so as to vigil their activities. The packet analyzer can then generate information that when the suspected users used the network with proper time and date.

PING

Ping is a computer network tool used to test whether a particular host is reachable across an IP network; it is also used to self test the network interface card of the computer, or as a speed test. It works by sending ICMP “echo request” packets to the target host and listening for ICMP “echo response” replies. Ping measures the round-trip time[1] and records any packet loss, and prints when finished a statistical summary of the echo response packets received, the minimum, mean, max and in some versions the standard deviation of the round trip time.The word ping is also frequently used as a verb or noun, where it is usually incorrectly used to refer to the round-trip time, or measuring the round-trip time. The tool is used in a simple denial-of-service attacks, known as a ping flood, in which the attacker overwhelms the victim with ICMP echo request packets. Several tools have been added and written to make IT the perfect choice to backup and restore whole partitions, an easy way.

TRACEROUTE

Traceroute is a computer network tool that shows us the route over the network between two systems, listing all the intermediate routers a connection must pass through to get to its destination. The traceroute tool is available on practically all Unix-like operating systems. Variants with similar functionality are also available, such as tracepath on modern Linux installations and tracert on Microsoft Windows operating systems. It helps us determine why our connections to a given server might be poor and helps us figure out where exactly the problem is. It also shows you how systems are connected to each other, letting you see how ISP connects to the Internet and how the target system is connected.

Implementation

Traceroute works by increasing the "time-to-live" value of each successive batch of packets sent. The first three packets sent have a time-to-live (TTL) value of one (implying that they are not forwarded by the next router and make only a single hop). The next three packets have a TTL value of 2, and so on. When a packet passes through a host, normally the host decrements the TTL value by one, and forwards the packet to the next host. When a packet with a TTL of one reaches a host, the host discards the packet and sends an ICMP time exceeded (type 11) packet to the sender. The traceroute utility uses these returning packets to produce a list of hosts that the packets have traversed en route to the destination. The three timestamp values returned for each host along the path are the delay values in milliseconds for each packet in the batch. If a packet does not return within the expected timeout window, a star is printed. Traceroute doesn't list the real hosts. It indicates that the first host is at one hop, the second host at two hops, etc. IP does not guarantee that all the packets take the same route. Also note that if the host at hop number N does not reply, the hop will be skipped in the output

.

Uses

Traceroute is used for network troubleshooting. By showing a list of routers traversed, it allows the user to identify the path taken to reach a particular destination on the network. It helps identifying routing problems and firewalls that may be blocking access to a site. It can be used to gather information about network infrastructure and IP ranges around a given host. It also tells us the IP addresses of the domains being used and the maximum number of hops it will take before it times out.

Security concerns

Traceroute has been frequently used by hackers to acquire sensitive information about company's network architectures and can quickly map out intermediate routers for known destinations on the architecture. So many networks block traceroute requests or de-prioritize the ICMP time exceeded messages. However, filtering traffic except at network end-points is a controversial practice.

MAC ADDRESS

A unique identifier assigned to network adapters by the manufacturer's for identification. it is intended to be permanent and globally unique; encoding manufacturer's registered identification number; but it is now possible to alter it.

IP ADDRESS

It is a unique number for identification of each system on internet. It might be permanently assigned or supplied each time a user connects to the internet, depending on the service provider, but it has only one IP address at a time.

PORT ADDRESSES

They are assigned at the beginning of the packet so as to make the monitoring of the data fast. However a machine can have more than one port addresses at a time, depending on the number of ports available.

CHAPTER 10- USES AND APPLICATIONS

USES

- Analyze network problems
- Detect network intrusion attempts
- Gain information for effecting a network intrusion
- Monitor network usage
- Gather and report network statistics
- Filter suspect content from network traffic
- Spy on other network users and collect sensitive information such as passwords (depending on any content encryption methods which may be in use)
- Reverse engineer proprietary protocols used over the network
- Debug client/server communications
- Debug network protocol implementations
- Can be used in education to demonstrate how network protocols work
- Often used in the development and debugging of networking software
- For a token ring network it can detect if the token has been lost or the presence of too many tokens
- Can detect if messages are being sent to a network adapter, if the network adapter did not report receiving the messages then this would localize the failure to the adapter
- Can detect excessive messages being sent by a port, detecting an error in the implementation
- Can collect statistics on the amount of traffic (number of messages) from a process detecting the need for more bandwidth or a better method
- Used to extract messages and reassemble into a complete form the traffic from a process, allowing it to be reverse engineered
- To analyze data sent to and from secure systems in order to understand and circumvent security measures, for the purposes of penetration testing or illegal activities

HOW DOES IT HELP

- Managing a LAN
- Creating network-oriented software
- Performing a security audit.
- Maintain efficient network data transmission
- Intrusion detection systems
- Identify problems with network-based applications.
- It costs a fraction of the price of information, time, software, and hardware that may potentially be lost or wasted by *not* using a network analyzer

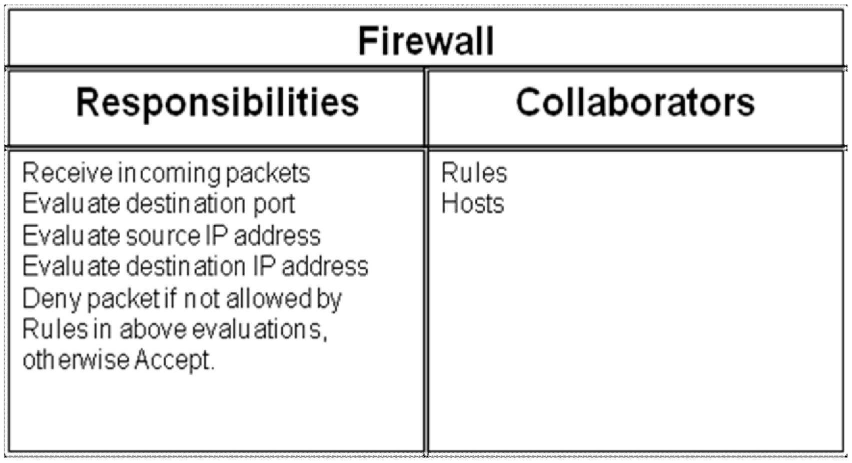
Chapter 11 : DESIGN

- After analysis and planning we created 3 main classes as
- Firewall
- Rules
- Host

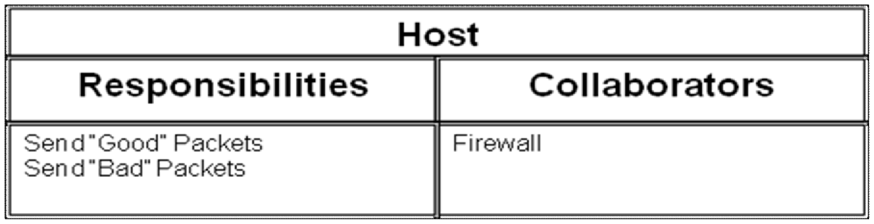
These classes were analyzed further with their relation and responsibilities. Unified Modeling Language (UML) was used in the analysis and design of this model.
We created a GUI (Graphic User Interface) for the to generate a "administrator" to "application" interaction interface. This GUI would be further utilized by the administrator to appropriately configure the network firewall system.
Furthermore, a "packet filteration" module was also generated, which would specifically be applied for the firewall application purposes with its main system tool being the "packet filteration process".

11.1 CRC Cards

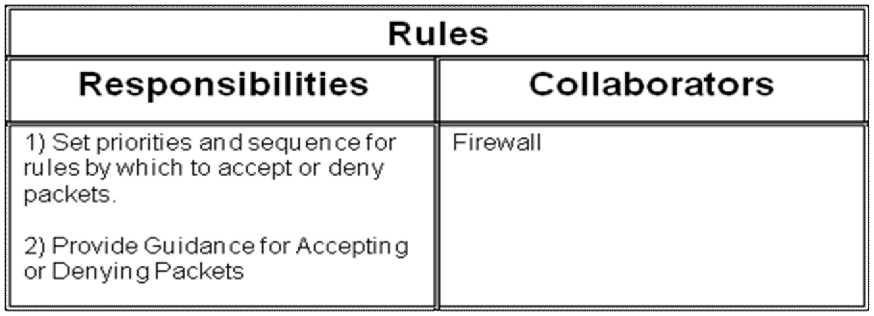
CRC, or class responsibility collaboration, cards are used to flesh out the characteristics of a class. This method provides and easy method for assigning particular behaviors and actions to objects.



Firewall	
Responsibilities	**Collaborators**
Receive incoming packets Evaluate destination port Evaluate source IP address Evaluate destination IP address Deny packet if not allowed by Rules in above evaluations, otherwise Accept.	Rules Hosts



Host	
Responsibilities	**Collaborators**
Send "Good" Packets Send "Bad" Packets	Firewall



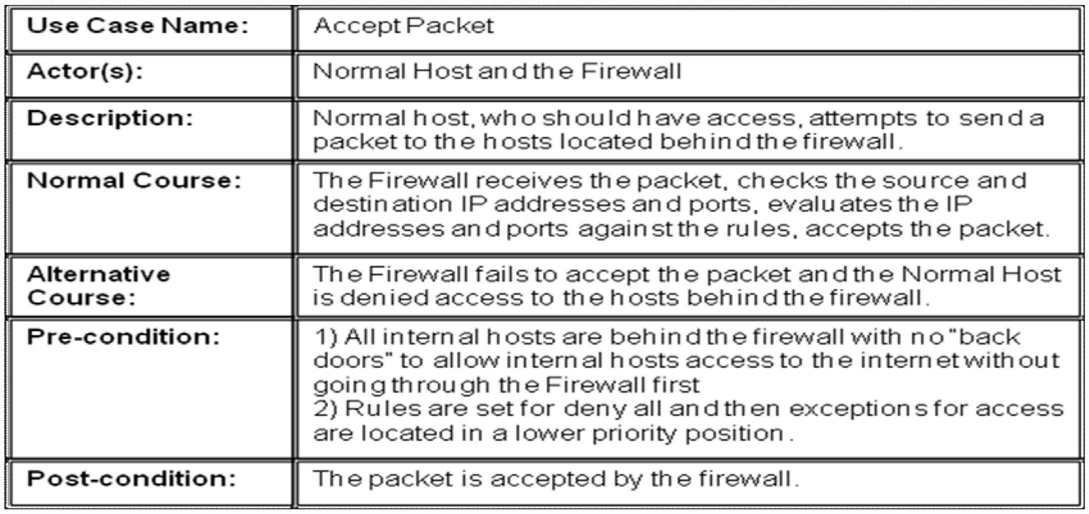
Rules	
Responsibilities	**Collaborators**
1) Set priorities and sequence for rules by which to accept or deny packets. 2) Provide Guidance for Accepting or Denying Packets	Firewall

11.2 Use Case Diagrams

In the Unified Modeling Language (UML) it is defined as a type of behavioral diagram defined by and created from a Use-case analysis. Its purpose is to present a graphical overview of the functionality provided by a system in terms of actors, their goals (represented as use cases), and any dependencies between those use cases.

The main purpose of a use case diagram is to show what system functions are performed for which actor. Roles of the actors in the system can be depicted.

11.2.1 Use case : Accept Packet

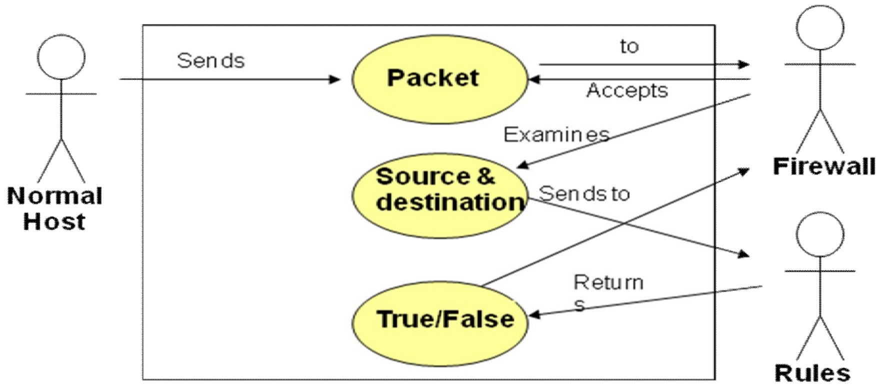


Use Case Name:	Accept Packet
Actor(s):	Normal Host and the Firewall
Description:	Normal host, who should have access, attempts to send a packet to the hosts located behind the firewall.
Normal Course:	The Firewall receives the packet, checks the source and destination IP addresses and ports, evaluates the IP addresses and ports against the rules, accepts the packet.
Alternative Course:	The Firewall fails to accept the packet and the Normal Host is denied access to the hosts behind the firewall.
Pre-condition:	1) All internal hosts are behind the firewall with no "back doors" to allow internal hosts access to the internet without going through the Firewall first 2) Rules are set for deny all and then exceptions for access are located in a lower priority position.
Post-condition:	The packet is accepted by the firewall.

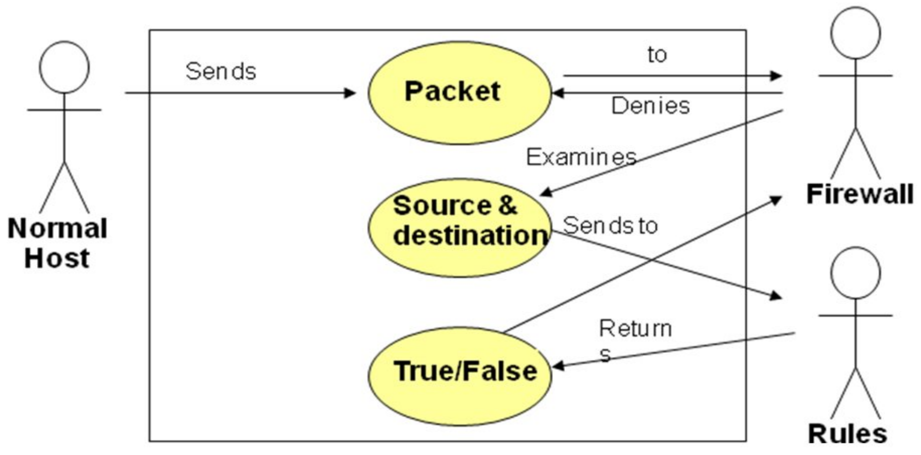
11.2.2 Use Case : Deny Packet

Actor(s):	Hacker Host and the Firewall
Description:	Hacker host, who should not have access, attempts to send a packet to the hosts located behind the firewall.
Normal Course:	The Firewall receives the packet, checks the source and destination IP addresses and ports, evaluates the IP addresses and ports against the rules, denies the packet.
Alternative Course:	The Firewall fails to deny the packet and the Hacker Host gains access to the hosts behind the firewall.
Pre-condition:	1) All internal hosts are behind the firewall with no "back doors" to allow internal hosts access to the internet without going through the Firewall first 2) Rules are set for deny all and then exceptions for access are located in a lower priority position.
Post-condition:	The packet is denied by the firewall.

11.2.3 Use Case Diagrams : Accept Packet



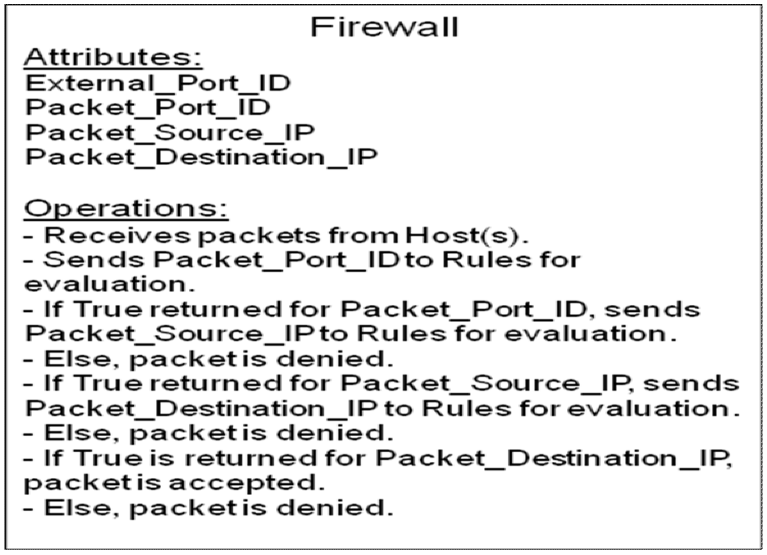
11.2.4 Use Case Diagram : Deny Packet



11.3 Class Diagrams

A class diagram in the Unified Modeling Language (UML) , is a type of static structure diagram that describes the structure of a system by showing the system's classes, their attributes, and the relationships between the classes.

11.3.1 Class Diagram : Firewall

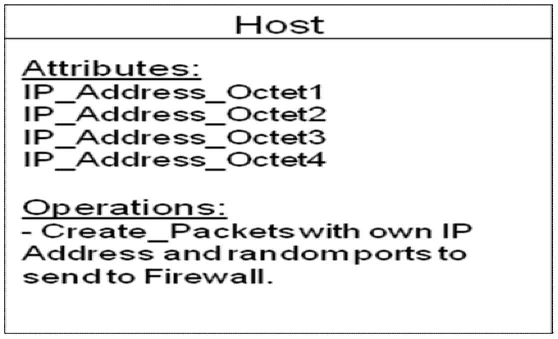


Firewall

Attributes:
External_Port_ID
Packet_Port_ID
Packet_Source_IP
Packet_Destination_IP

Operations:
- Receives packets from Host(s).
- Sends Packet_Port_ID to Rules for evaluation.
- If True returned for Packet_Port_ID, sends Packet_Source_IP to Rules for evaluation.
- Else, packet is denied.
- If True returned for Packet_Source_IP, sends Packet_Destination_IP to Rules for evaluation.
- Else, packet is denied.
- If True is returned for Packet_Destination_IP, packet is accepted.
- Else, packet is denied.

11.3.2 Class Diagram : Host

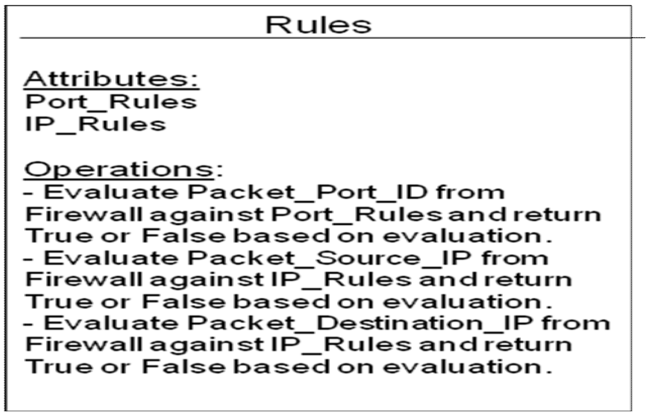


Host

Attributes:
IP_Address_Octet1
IP_Address_Octet2
IP_Address_Octet3
IP_Address_Octet4

Operations:
- Create_Packets with own IP Address and random ports to send to Firewall.

11.3.3 Class Diagram : Rules

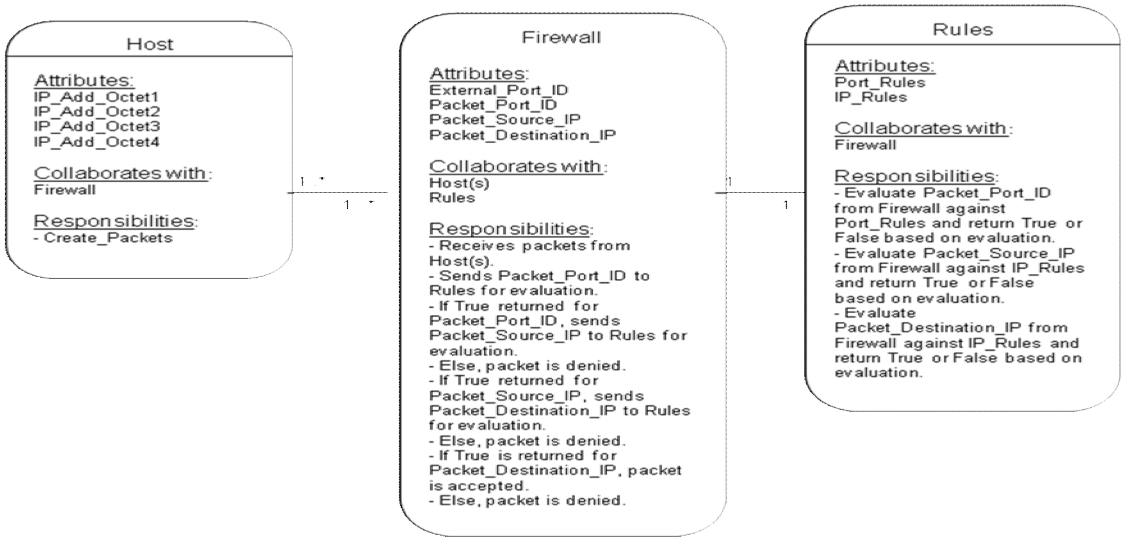

Rules

Attributes:
Port_Rules
IP_Rules

Operations:
- Evaluate Packet_Port_ID from Firewall against Port_Rules and return True or False based on evaluation.
- Evaluate Packet_Source_IP from Firewall against IP_Rules and return True or False based on evaluation.
- Evaluate Packet_Destination_IP from Firewall against IP_Rules and return True or False based on evaluation.

11.4 Collaboration Diagram

Collaboration diagram, also called a communication diagram or interaction diagram, is an illustration of the relationships and interactions among software objects in the Unified Modeling Language (UML). The concept is more than a decade old although it has been refined as modeling paradigms have evolved. A collaboration diagram resembles a flowchart that portrays the roles, functionality and behavior of individual objects as well as the overall operation of the system in real time. Objects are shown as rectangles with naming labels inside. These labels are preceded by colons and may be underlined. The relationships between the objects are shown as lines connecting the rectangles. The messages between objects are shown as arrows connecting the relevant rectangles along with labels that define the message sequencing.



Host

Attributes:
IP_Add_Octet1
IP_Add_Octet2
IP_Add_Octet3
IP_Add_Octet4

Collaborates with:
Firewall

Responsibilities:
- Create_Packets

Firewall

Attributes:
External_Port_ID
Packet_Port_ID
Packet_Source_IP
Packet_Destination_IP

Collaborates with:
Host(s)
Rules

Responsibilities:
- Receives packets from Host(s).
- Sends Packet_Port_ID to Rules for evaluation.
- If True returned for Packet_Port_ID, sends Packet_Source_IP to Rules for evaluation.
- Else, packet is denied.
- If True returned for Packet_Source_IP, sends Packet_Destination_IP to Rules for evaluation.
- Else, packet is denied.
- If True is returned for Packet_Destination_IP, packet is accepted.
- Else, packet is denied.

Rules

Attributes:
Port_Rules
IP_Rules

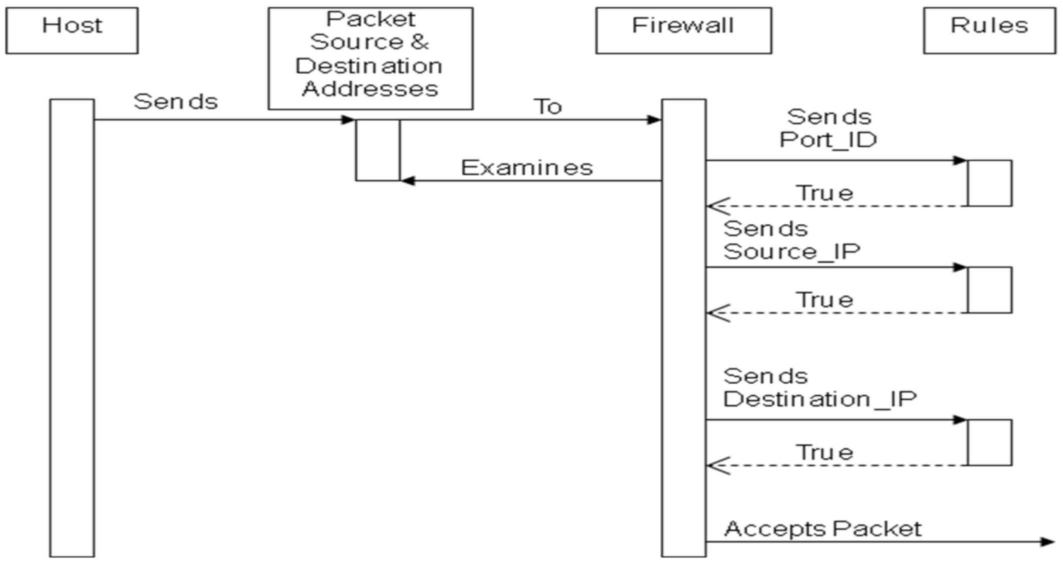
Collaborates with:
Firewall

Responsibilities:
- Evaluate Packet_Port_ID from Firewall against Port_Rules and return True or False based on evaluation.
- Evaluate Packet_Source_IP from Firewall against IP_Rules and return True or False based on evaluation.
- Evaluate Packet_Destination_IP from Firewall against IP_Rules and return True or False based on evaluation.

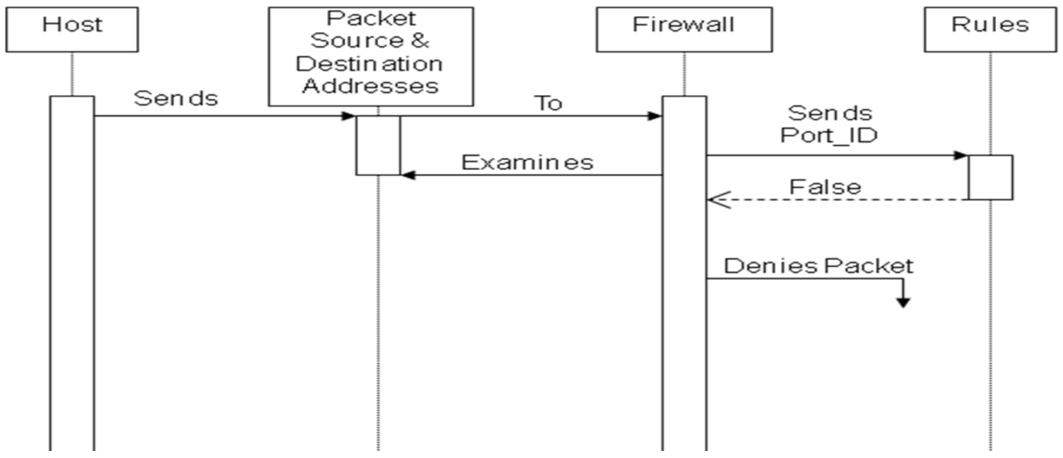
11.5 Sequence Diagram

A sequence diagram in Unified Modelling Language (UML) is a kind of interaction diagram that shows how processes operate with one another and in what order. It is a construct of a Message Sequence Chart. Sequence diagrams are sometimes called Event-trace diagrams, event scenarios, and timing diagrams.

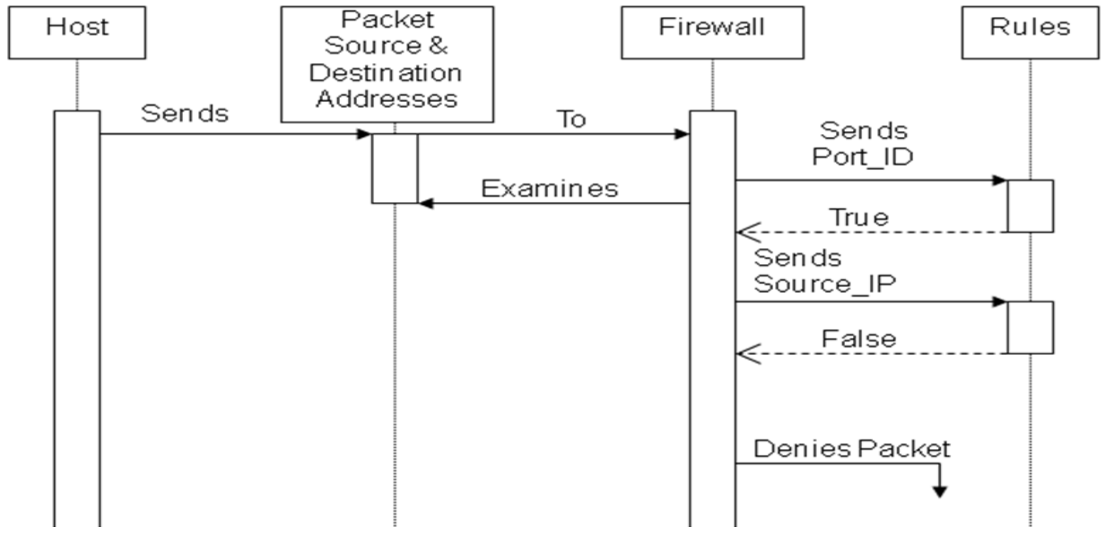
11.5.1 Sequence Diagram : Accept Packet



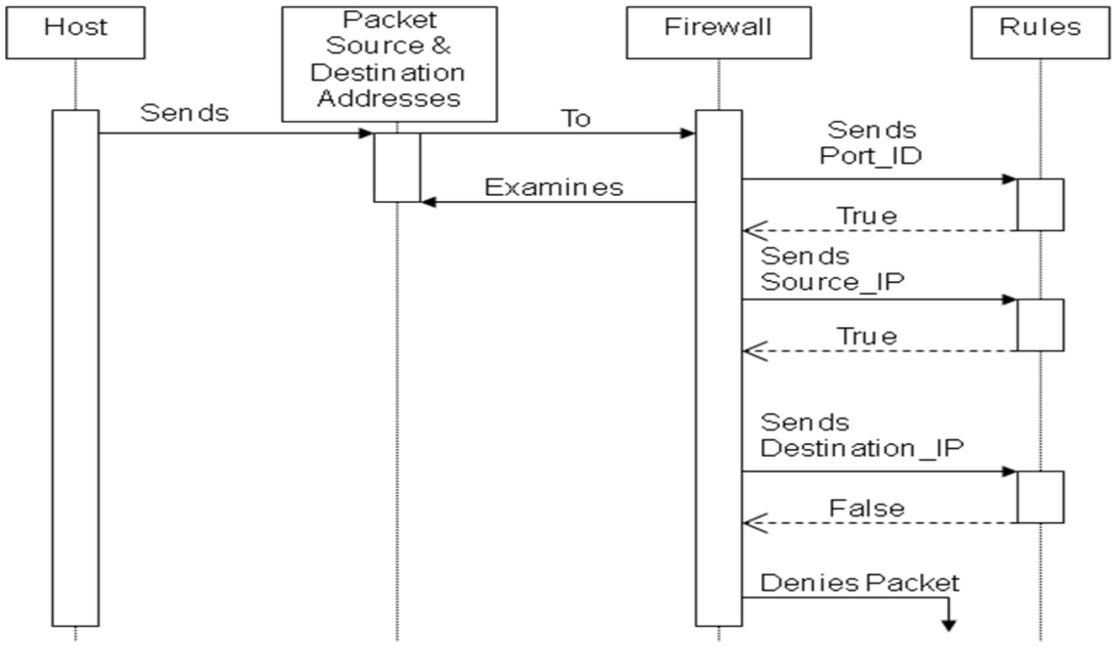
11.5.2 Sequence Diagram : Deny Packet(port)



11.5.3 Sequence Diagram : Deny Packet (Source IP address)

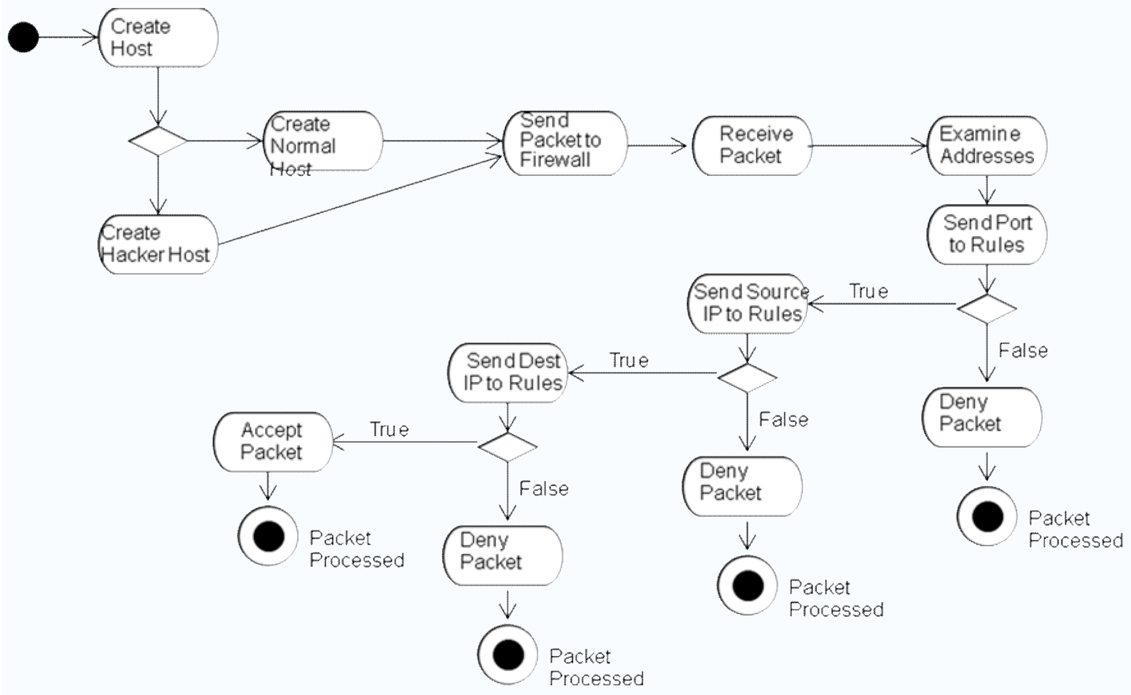


11.5.4 Sequence Diagram : Deny Packet(Destination IP address)



11.6 Activity Diagram

Activity diagrams are a loosely defined diagram technique for showing workflows of stepwise activities and actions, with support for choice, iteration and concurrency. In the Unified Modeling Language, activity diagrams can be used to describe the business and operational step-by-step workflows of components in a system. An activity diagram shows the overall flow of control.

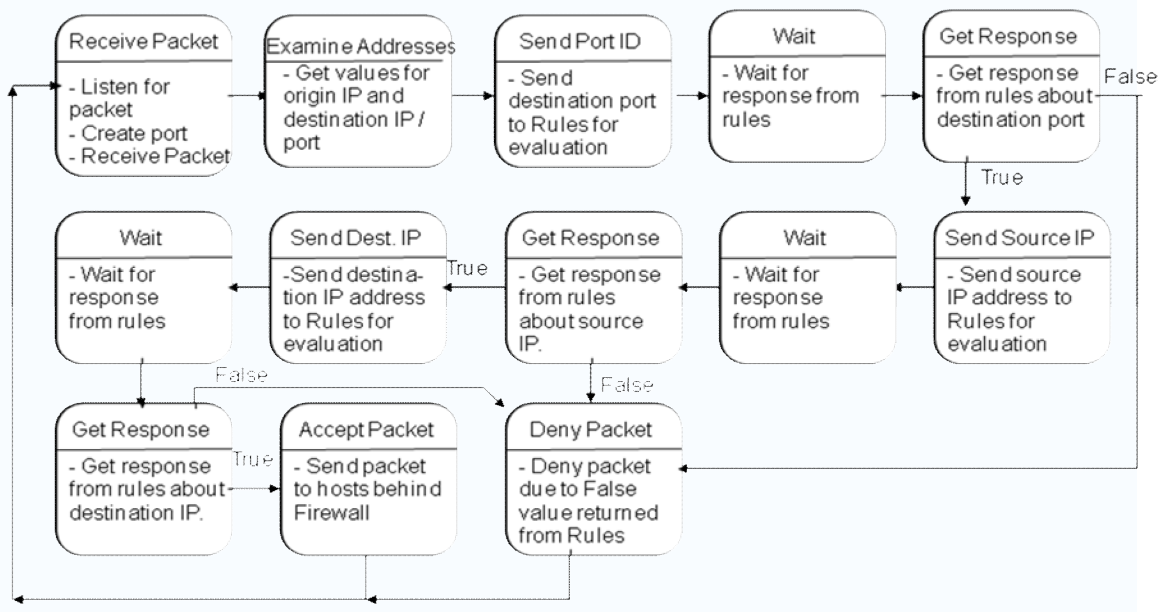


Firewall activity diagram

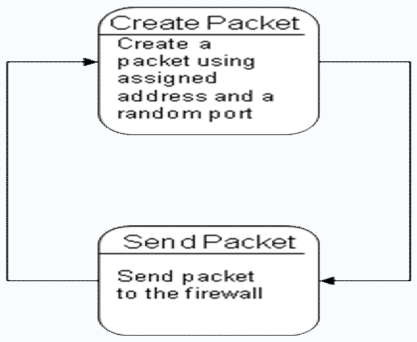
11.7 State Diagram

A state diagram is a type of diagram used in computer science and related fields to describe the behavior of systems. State diagrams require that the system described is composed of a finite number of states; sometimes, this is indeed the case, while at other times this is a reasonable abstraction. There are many forms of state diagrams, which differ slightly and have different semantics.

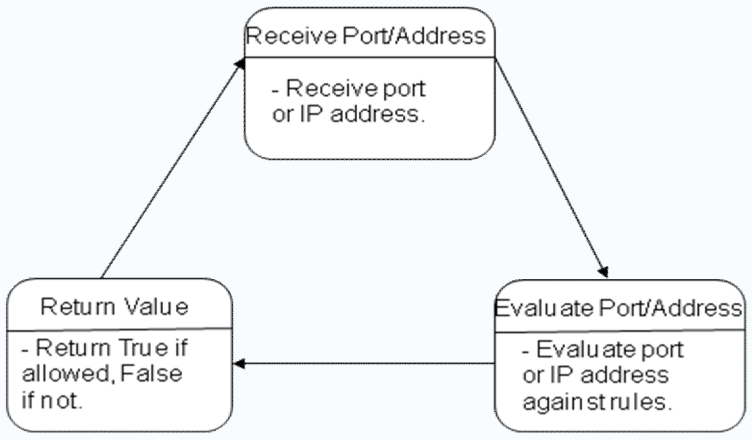
11.7.1 State Diagram : Firewall


Receive Packet
- Listen for packet
- Create port
- Receive Packet
Examine Addresses
- Get values for origin IP and destination IP / port
Send Port ID
- Send destination port to Rules for evaluation
Wait
- Wait for response from rules
Get Response
- Get response from rules about destination port
False
True
Send Source IP
- Send source IP address to Rules for evaluation
Wait
- Wait for response from rules
Get Response
- Get response from rules about source IP.
True
Send Dest. IP
-Send destination IP address to Rules for evaluation
Wait
- Wait for response from rules
False
False
Get Response
- Get response from rules about destination IP.
True
Accept Packet
- Send packet to hosts behind Firewall
Deny Packet
- Deny packet due to False value returned from Rules

11.7.2 State Diagram : Host


Create Packet
Create a packet using assigned address and a random port
Send Packet
Send packet to the firewall

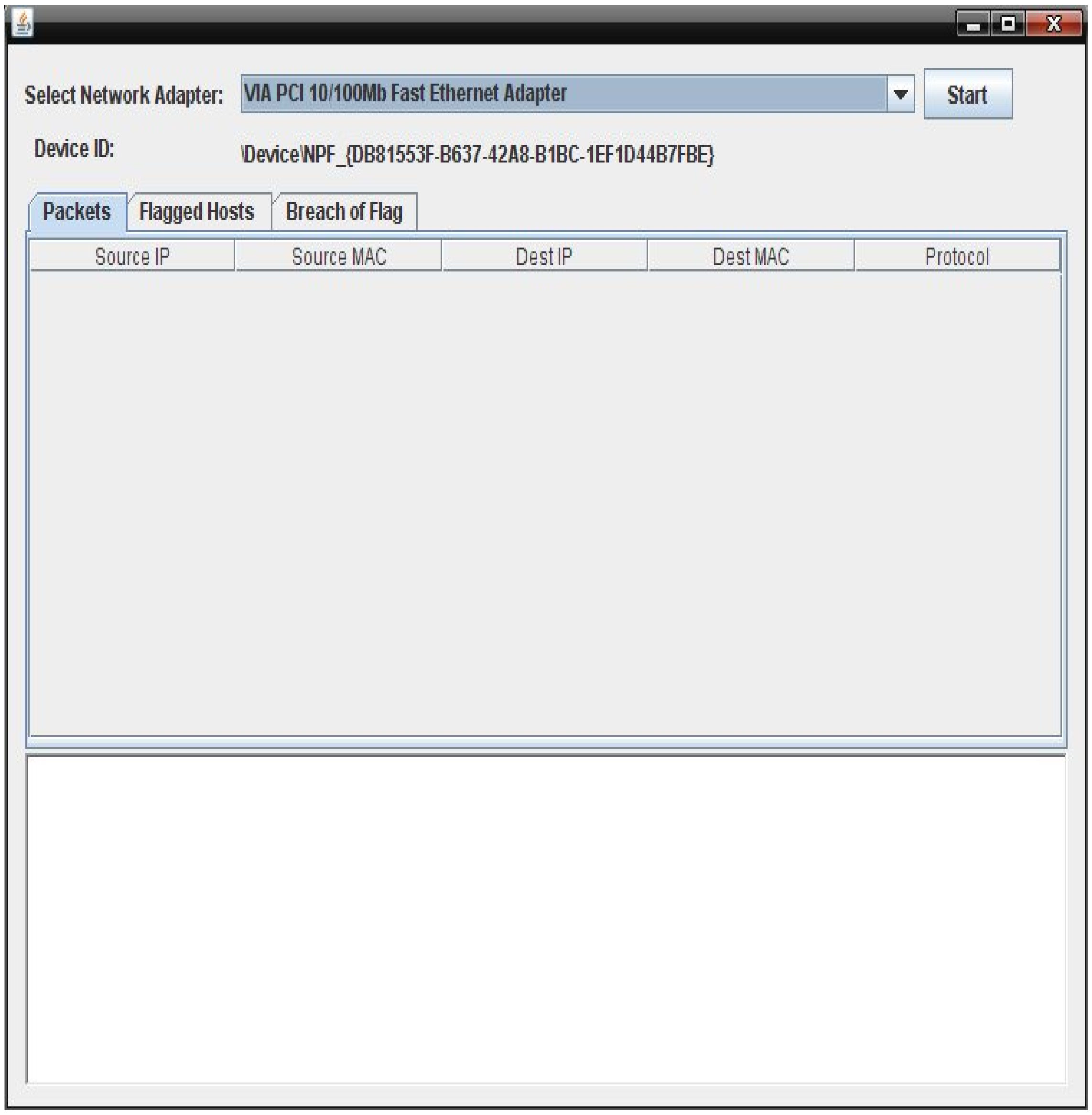
11.7.3 State Diagram : Rules


Receive Port/Address
- Receive port or IP address.
Evaluate Port/Address
- Evaluate port or IP address against rules.
Return Value
- Return True if allowed, False if not.

Chapter 12: GRAPHICAL USER INTERFACE(GUI)

12.1 Design

Design of the graphical user interface was plotted in Netbeans 6.0. NetBeans IDE, the full-featured integrated environment for software developers and NetBeans Platform, the widely adopted infrastructure backplane for complex desktop applications.

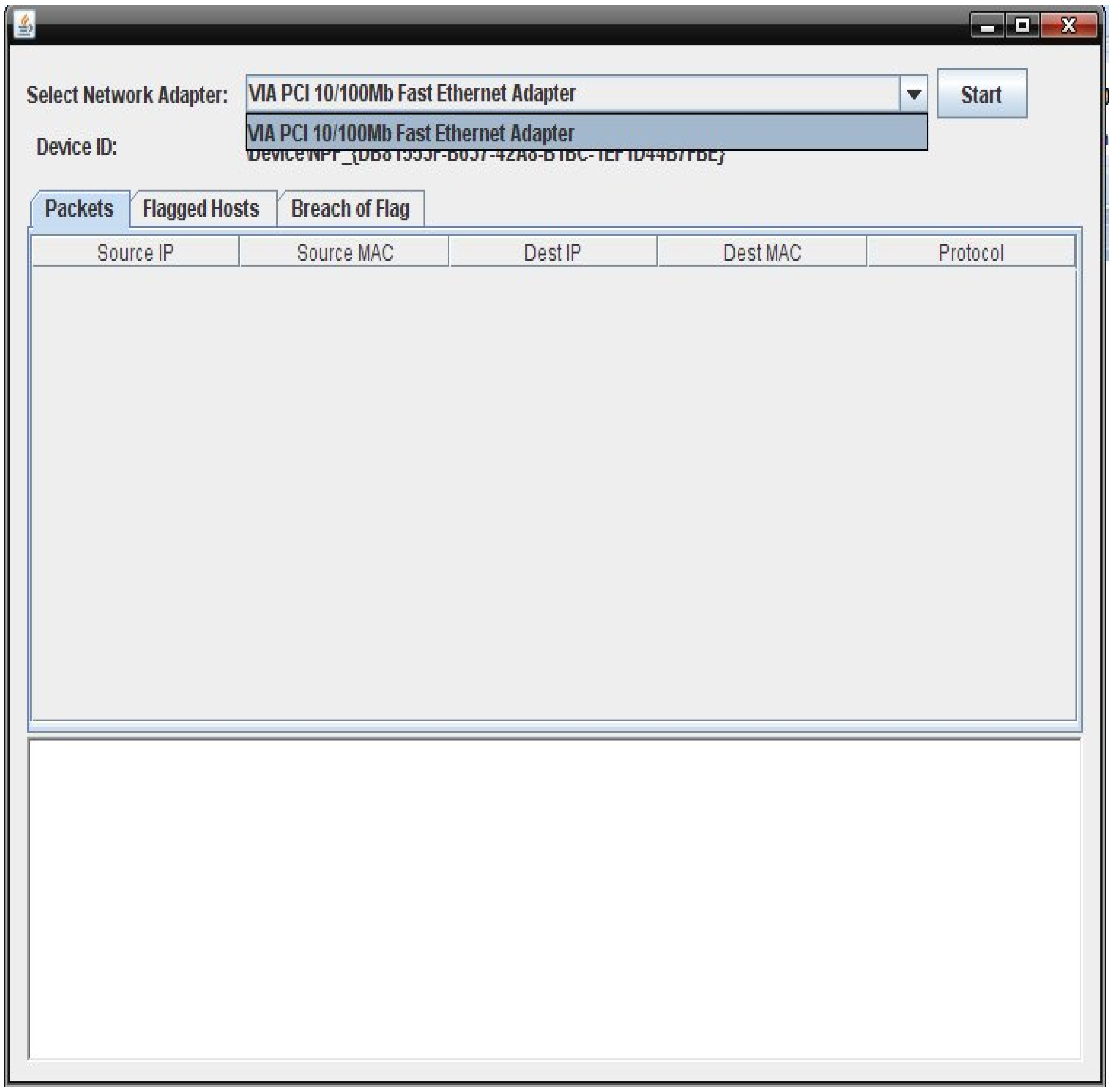


Select Network Adapter: VIA PCI 10/100Mb Fast Ethernet Adapter

Device ID: \Device\NPF_{DB81553F-B637-42A8-B1BC-1EF1D44B7FBE}

12.2 Screen shots of the working GUI

12.2.1 Screen shot: Selection of the Network Adapter

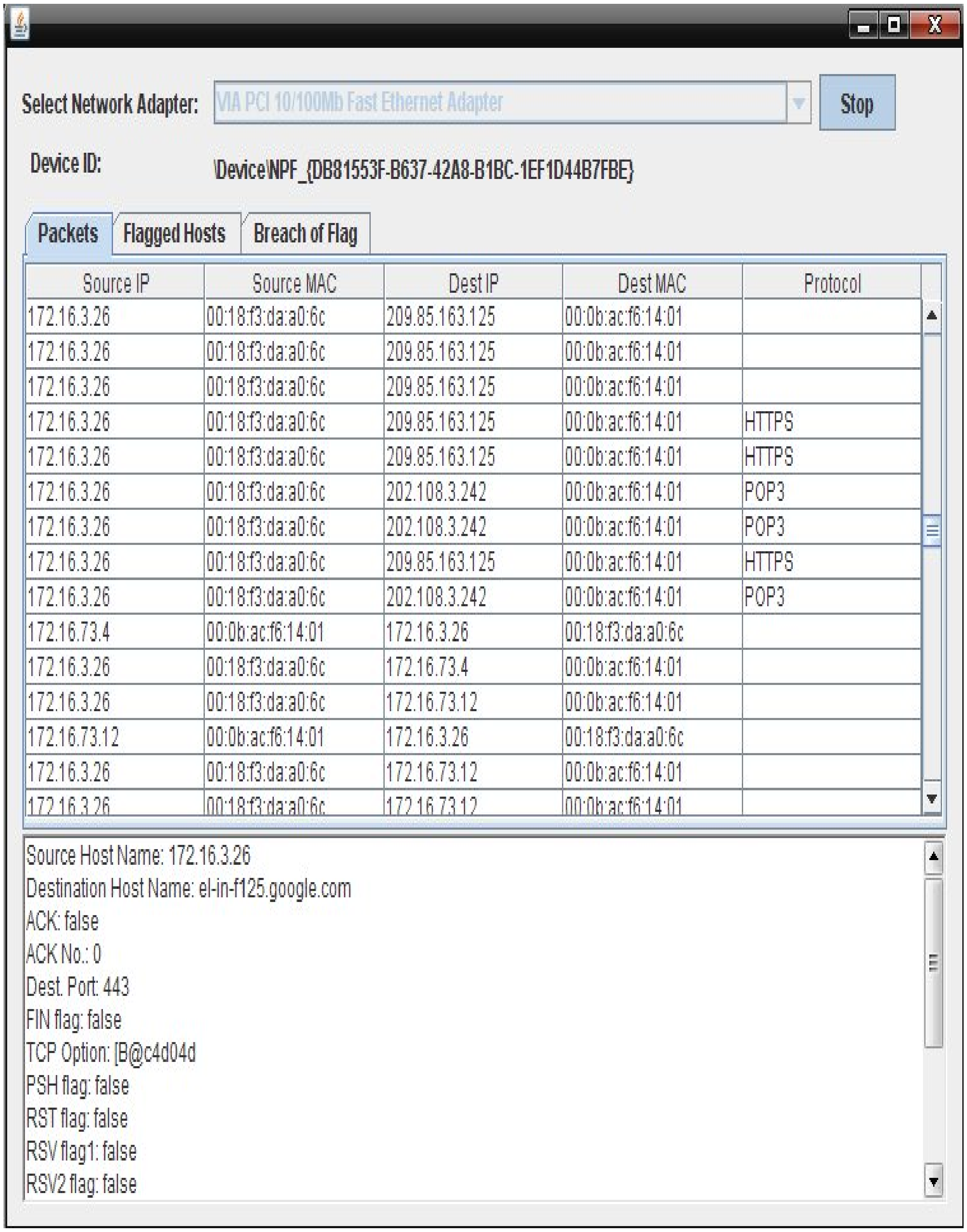

Select Network Adapter: VIA PCI 10/100Mb Fast Ethernet Adapter

VIA PCI 10/100Mb Fast Ethernet Adapter

Device ID:

Source IP	Source MAC	Dest IP	Dest MAC	Protocol

12.2.2 Screen Shot: Screen shots for the network monitor

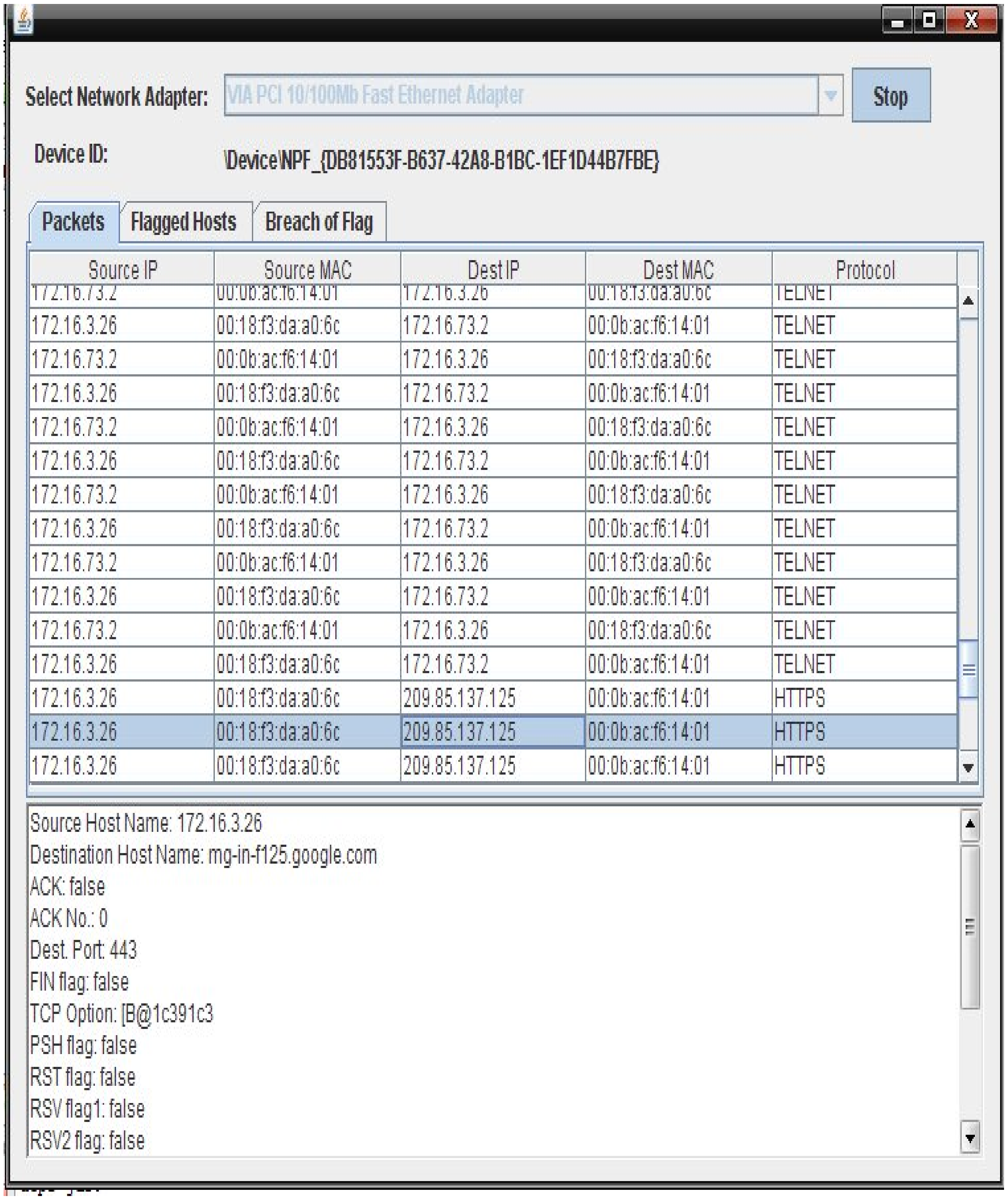


Device ID: \Device\NPF_{DB81553F-B637-42A8-B1BC-1EF1D44B7FBE}

Source IP	Source MAC	Dest IP	Dest MAC	Protocol
172.16.3.26	00:18:f3:da:a0:6c	209.85.163.125	00:0b:ac:f6:14:01	
172.16.3.26	00:18:f3:da:a0:6c	209.85.163.125	00:0b:ac:f6:14:01	
172.16.3.26	00:18:f3:da:a0:6c	209.85.163.125	00:0b:ac:f6:14:01	
172.16.3.26	00:18:f3:da:a0:6c	209.85.163.125	00:0b:ac:f6:14:01	HTTPS
172.16.3.26	00:18:f3:da:a0:6c	209.85.163.125	00:0b:ac:f6:14:01	HTTPS
172.16.3.26	00:18:f3:da:a0:6c	202.108.3.242	00:0b:ac:f6:14:01	POP3
172.16.3.26	00:18:f3:da:a0:6c	202.108.3.242	00:0b:ac:f6:14:01	POP3
172.16.3.26	00:18:f3:da:a0:6c	209.85.163.125	00:0b:ac:f6:14:01	HTTPS
172.16.3.26	00:18:f3:da:a0:6c	202.108.3.242	00:0b:ac:f6:14:01	POP3
172.16.73.4	00:0b:ac:f6:14:01	172.16.3.26	00:18:f3:da:a0:6c	
172.16.3.26	00:18:f3:da:a0:6c	172.16.73.4	00:0b:ac:f6:14:01	
172.16.3.26	00:18:f3:da:a0:6c	172.16.73.12	00:0b:ac:f6:14:01	
172.16.73.12	00:0b:ac:f6:14:01	172.16.3.26	00:18:f3:da:a0:6c	
172.16.3.26	00:18:f3:da:a0:6c	172.16.73.12	00:0b:ac:f6:14:01	
172.16.3.26	00:18:f3:da:a0:6c	172.16.73.12	00:0b:ac:f6:14:01	

```
Source Host Name: 172.16.3.26
Destination Host Name: el-in-f125.google.com
ACK: false
ACK No.: 0
Dest. Port: 443
FIN flag: false
TCP Option: [B@c4d04d
PSH flag: false
RST flag: false
RSV flag1: false
RSV2 flag: false
```

12.2.2 Screen Shot: Screen shots for the TCP packet details being generated with identification of the source and destination IP addresses and the port

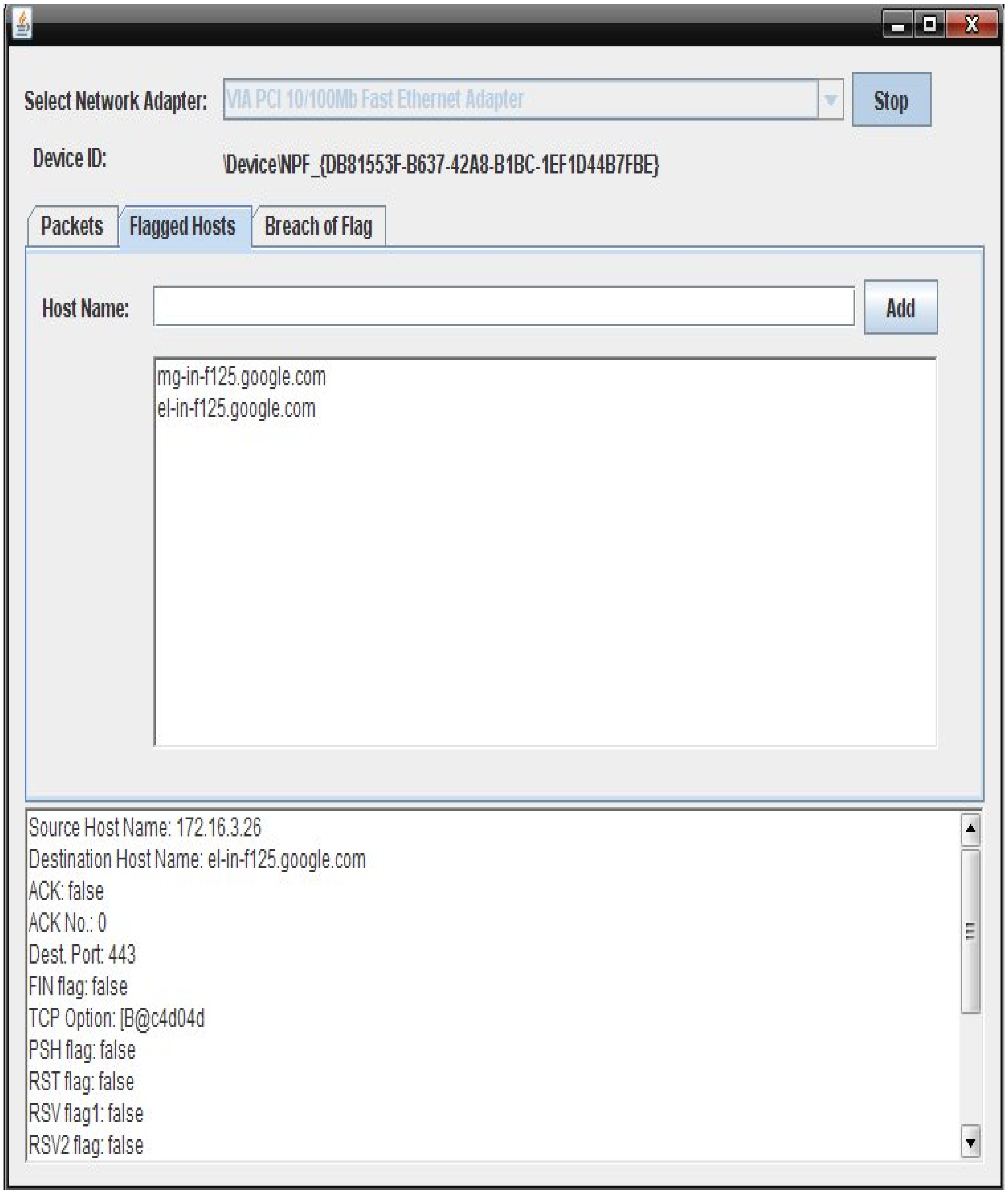


Device ID: \Device\NPF_{DB81553F-B637-42A8-B1BC-1EF1D44B7FBE}

Source IP	Source MAC	Dest IP	Dest MAC	Protocol
172.16.73.2	00:0b:ac:f6:14:01	172.16.3.26	00:18:f3:da:a0:6c	TELNET
172.16.3.26	00:18:f3:da:a0:6c	172.16.73.2	00:0b:ac:f6:14:01	TELNET
172.16.73.2	00:0b:ac:f6:14:01	172.16.3.26	00:18:f3:da:a0:6c	TELNET
172.16.3.26	00:18:f3:da:a0:6c	172.16.73.2	00:0b:ac:f6:14:01	TELNET
172.16.73.2	00:0b:ac:f6:14:01	172.16.3.26	00:18:f3:da:a0:6c	TELNET
172.16.3.26	00:18:f3:da:a0:6c	172.16.73.2	00:0b:ac:f6:14:01	TELNET
172.16.73.2	00:0b:ac:f6:14:01	172.16.3.26	00:18:f3:da:a0:6c	TELNET
172.16.3.26	00:18:f3:da:a0:6c	172.16.73.2	00:0b:ac:f6:14:01	TELNET
172.16.73.2	00:0b:ac:f6:14:01	172.16.3.26	00:18:f3:da:a0:6c	TELNET
172.16.3.26	00:18:f3:da:a0:6c	172.16.73.2	00:0b:ac:f6:14:01	TELNET
172.16.73.2	00:0b:ac:f6:14:01	172.16.3.26	00:18:f3:da:a0:6c	TELNET
172.16.3.26	00:18:f3:da:a0:6c	172.16.73.2	00:0b:ac:f6:14:01	TELNET
172.16.3.26	00:18:f3:da:a0:6c	209.85.137.125	00:0b:ac:f6:14:01	HTTPS
172.16.3.26	00:18:f3:da:a0:6c	209.85.137.125	00:0b:ac:f6:14:01	HTTPS
172.16.3.26	00:18:f3:da:a0:6c	209.85.137.125	00:0b:ac:f6:14:01	HTTPS

```
Source Host Name: 172.16.3.26
Destination Host Name: mg-in-f125.google.com
ACK: false
ACK No.: 0
Dest. Port: 443
FIN flag: false
TCP Option: [B@1c391c3
PSH flag: false
RST flag: false
RSV flag1: false
RSV2 flag: false
```

12.2.3 Screen Shot: Screen shots for the Flagged Host

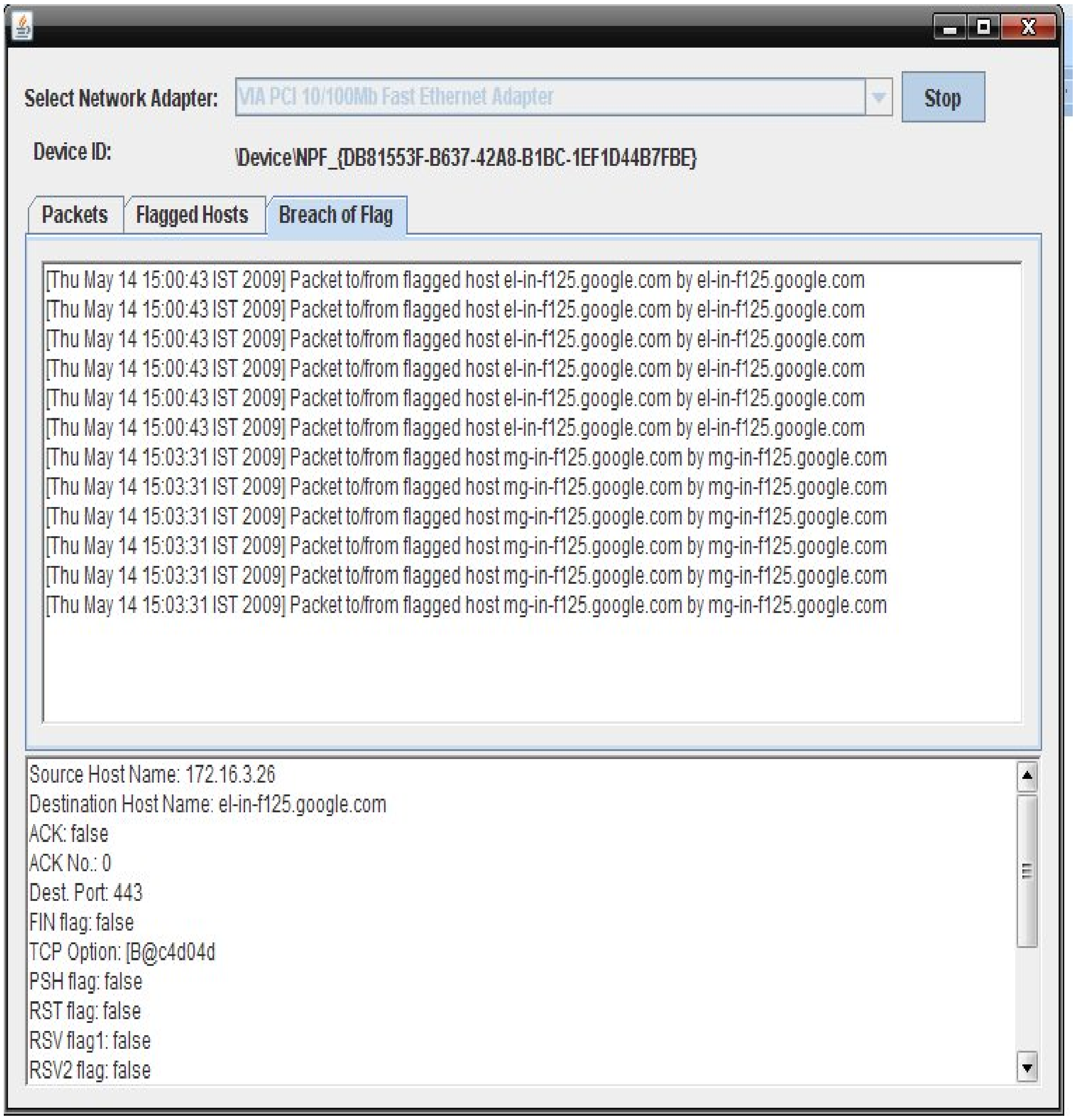


Device ID: \Device\NPF_{DB81553F-B637-42A8-B1BC-1EF1D44B7FBE}

```
mg-in-f125.google.com
el-in-f125.google.com
```

```
Source Host Name: 172.16.3.26
Destination Host Name: el-in-f125.google.com
ACK: false
ACK No.: 0
Dest. Port: 443
FIN flag: false
TCP Option: [B@c4d04d
PSH flag: false
RST flag: false
RSV flag1: false
RSV2 flag: false
```

12.2.4 Screen Shot: Screen shots for the Breaching of the flagged host



Device ID: \Device\NPF_{DB81553F-B637-42A8-B1BC-1EF1D44B7FBE}

```
[Thu May 14 15:00:43 IST 2009] Packet to/from flagged host el-in-f125.google.com by el-in-f125.google.com
[Thu May 14 15:00:43 IST 2009] Packet to/from flagged host el-in-f125.google.com by el-in-f125.google.com
[Thu May 14 15:00:43 IST 2009] Packet to/from flagged host el-in-f125.google.com by el-in-f125.google.com
[Thu May 14 15:00:43 IST 2009] Packet to/from flagged host el-in-f125.google.com by el-in-f125.google.com
[Thu May 14 15:00:43 IST 2009] Packet to/from flagged host el-in-f125.google.com by el-in-f125.google.com
[Thu May 14 15:00:43 IST 2009] Packet to/from flagged host el-in-f125.google.com by el-in-f125.google.com
[Thu May 14 15:03:31 IST 2009] Packet to/from flagged host mg-in-f125.google.com by mg-in-f125.google.com
[Thu May 14 15:03:31 IST 2009] Packet to/from flagged host mg-in-f125.google.com by mg-in-f125.google.com
[Thu May 14 15:03:31 IST 2009] Packet to/from flagged host mg-in-f125.google.com by mg-in-f125.google.com
[Thu May 14 15:03:31 IST 2009] Packet to/from flagged host mg-in-f125.google.com by mg-in-f125.google.com
[Thu May 14 15:03:31 IST 2009] Packet to/from flagged host mg-in-f125.google.com by mg-in-f125.google.com
[Thu May 14 15:03:31 IST 2009] Packet to/from flagged host mg-in-f125.google.com by mg-in-f125.google.com
```

```
Source Host Name: 172.16.3.26
Destination Host Name: el-in-f125.google.com
ACK: false
ACK No.: 0
Dest. Port: 443
FIN flag: false
TCP Option: [B@c4d04d
PSH flag: false
RST flag: false
RSV flag1: false
RSV2 flag: false
```

Chapter 13: CODING

```
package firewall;
import java.io.IOException;
import jpcap.*;
import jpcap.packet.*;
import java.util.*;
import javax.swing.table.*;


 class frmMain extends javax.swing.JFrame {
    NetworkInterface[] devices=JpcapCaptor.getDeviceList();
    packetReader pr;
    Vector tcpPackets=new Vector();
    /** Creates new form frmMain */
    public frmMain() {
        initComponents();
        getNetworkAdapters();
    }

    void getNetworkAdapters(){
        //adding each device to the list
        for(int i=0;i<devices.length;i++){
            cmbAdapters.addItem(devices[i].description);
        }
    }

    /** This method is called from within the constructor to
     * initialize the form.
     * WARNING: Do NOT modify this code. The content of this method is
     * always regenerated by the Form Editor.
     */
    //    <editor-fold    defaultstate="collapsed"    desc="Generated    Code">//GEN-
BEGIN:initComponents
    private void initComponents() {
```

```
jLabel1 = new javax.swing.JLabel();
cmbAdapters = new javax.swing.JComboBox();
jLabel2 = new javax.swing.JLabel();
lblDeviceName = new javax.swing.JLabel();
lstPackets = new java.awt.List();
cmdStart2 = new javax.swing.JToggleButton();
jTabbedPane1 = new javax.swing.JTabbedPane();
jPanel1 = new javax.swing.JPanel();
jScrollPane1 = new javax.swing.JScrollPane();
tblTCP = new javax.swing.JTable();
jPanel2 = new javax.swing.JPanel();
jLabel3 = new javax.swing.JLabel();
txtHostName = new javax.swing.JTextField();
lstHosts = new java.awt.List();
cmdAdd = new javax.swing.JButton();
jPanel3 = new javax.swing.JPanel();
lstBreach = new java.awt.List();

setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);

jLabel1.setText("Select Network Adapter:");

cmbAdapters.addItemListener(new java.awt.event.ItemListener() {
    public void itemStateChanged(java.awt.event.ItemEvent evt) {
        cmbAdaptersItemStateChanged(evt);
    }
});
cmbAdapters.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        cmbAdaptersActionPerformed(evt);
    }
});
```

```
jLabel2.setText("Device ID:");

cmdStart2.setText("Start");
cmdStart2.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        cmdStart2ActionPerformed(evt);
    }
});

tblTCP.setModel(new javax.swing.table.DefaultTableModel(
    new Object [][] {

    },
    new String [] {
        "Source IP", "Source MAC", "Dest IP", "Dest MAC", "Protocol"
    }
) {
    boolean[] canEdit = new boolean [] {
        false, false, false, false, false
    };

    public boolean isCellEditable(int rowIndex, int columnIndex) {
        return canEdit [columnIndex];
    }
});
tblTCP.addMouseListener(new java.awt.event.MouseAdapter() {
    public void mouseClicked(java.awt.event.MouseEvent evt) {
        tblTCPMouseClicked(evt);
    }
});
tblTCP.addKeyListener(new java.awt.event.KeyAdapter() {
    public void keyPressed(java.awt.event.KeyEvent evt) {
        tblTCPKeyPressed(evt);
    }
```

```
        });
        jScrollPane1.setViewportView(tblTCP);

        javax.swing.GroupLayout jPanel1Layout = new javax.swing.GroupLayout(jPanel1);
        jPanel1.setLayout(jPanel1Layout);
        jPanel1Layout.setHorizontalGroup(
            jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addComponent(jScrollPane1,     javax.swing.GroupLayout.DEFAULT_SIZE,     694,
Short.MAX_VALUE)
        );
        jPanel1Layout.setVerticalGroup(
            jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(jPanel1Layout.createSequentialGroup()
                .addComponent(jScrollPane1, javax.swing.GroupLayout.PREFERRED_SIZE, 254,
javax.swing.GroupLayout.PREFERRED_SIZE)
                .addContainerGap(javax.swing.GroupLayout.DEFAULT_SIZE,
Short.MAX_VALUE))
        );

        jTabbedPane1.addTab("Packets", jPanel1);

        jLabel3.setText("Host Name:");

        txtHostName.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                txtHostNameActionPerformed(evt);
            }
        });

        cmdAdd.setText("Add");
        cmdAdd.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                cmdAddActionPerformed(evt);
            }
        });

        javax.swing.GroupLayout jPanel2Layout = new javax.swing.GroupLayout(jPanel2);
        jPanel2.setLayout(jPanel2Layout);
        jPanel2Layout.setHorizontalGroup(
            jPanel2Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(jPanel2Layout.createSequentialGroup()
                .addContainerGap()
                .addComponent(jLabel3)
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)

.addGroup(jPanel2Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADI
NG)
                    .addGroup(javax.swing.GroupLayout.Alignment.TRAILING,
jPanel2Layout.createSequentialGroup()
                        .addComponent(txtHostName,     javax.swing.GroupLayout.DEFAULT_SIZE,
529, Short.MAX_VALUE)
                        .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                        .addComponent(cmdAdd))
                    .addComponent(lstHosts,     javax.swing.GroupLayout.DEFAULT_SIZE,     586,
Short.MAX_VALUE))
                .addGap(32, 32, 32))
        );
        jPanel2Layout.setVerticalGroup(
            jPanel2Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(jPanel2Layout.createSequentialGroup()
                .addContainerGap()

.addGroup(jPanel2Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASEL
INE)
                    .addComponent(jLabel3)
                    .addComponent(cmdAdd)
                .addComponent(txtHostName,    javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE))
            .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
            .addComponent(lstHosts,    javax.swing.GroupLayout.PREFERRED_SIZE,    186,
javax.swing.GroupLayout.PREFERRED_SIZE)
            .addContainerGap(26, Short.MAX_VALUE))
    );

    jTabbedPane1.addTab("Flagged Hosts", jPanel2);

    lstBreach.addActionListener(new java.awt.event.ActionListener() {
      public void actionPerformed(java.awt.event.ActionEvent evt) {
        lstBreachActionPerformed(evt);
      }
    });

    javax.swing.GroupLayout jPanel3Layout = new javax.swing.GroupLayout(jPanel3);
    jPanel3.setLayout(jPanel3Layout);
    jPanel3Layout.setHorizontalGroup(
      jPanel3Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
      .addGroup(jPanel3Layout.createSequentialGroup()
        .addContainerGap()
        .addComponent(lstBreach,    javax.swing.GroupLayout.DEFAULT_SIZE,    674,
Short.MAX_VALUE)
        .addContainerGap())
    );
    jPanel3Layout.setVerticalGroup(
      jPanel3Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
      .addGroup(jPanel3Layout.createSequentialGroup()
        .addContainerGap()
        .addComponent(lstBreach,    javax.swing.GroupLayout.DEFAULT_SIZE,    236,
Short.MAX_VALUE)
        .addContainerGap())
        );

        jTabbedPane1.addTab("Breach of Flag", jPanel3);

        javax.swing.GroupLayout layout = new javax.swing.GroupLayout(getContentPane());
        getContentPane().setLayout(layout);
        layout.setHorizontalGroup(
            layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(javax.swing.GroupLayout.Alignment.TRAILING,
layout.createSequentialGroup()
                .addContainerGap()

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.TRAILING)
                    .addGroup(javax.swing.GroupLayout.Alignment.LEADING,
layout.createSequentialGroup()

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                            .addComponent(jLabel1)
                            .addGroup(layout.createSequentialGroup()
                                .addGap(7, 7, 7)
                                .addComponent(jLabel2)))
                        .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                            .addGroup(layout.createSequentialGroup()
                                .addComponent(lblDeviceName,
javax.swing.GroupLayout.DEFAULT_SIZE, 525, Short.MAX_VALUE)
                                .addGap(50, 50, 50))
                            .addGroup(layout.createSequentialGroup()
                                .addComponent(cmbAdapters,
javax.swing.GroupLayout.PREFERRED_SIZE,                                          469,
javax.swing.GroupLayout.PREFERRED_SIZE)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
```

```
                        .addComponent(cmdStart2))))
                    .addComponent(jTabbedPane1,
javax.swing.GroupLayout.Alignment.LEADING,
javax.swing.GroupLayout.DEFAULT_SIZE, 699, Short.MAX_VALUE)
                    .addComponent(lstPackets,       javax.swing.GroupLayout.Alignment.LEADING,
javax.swing.GroupLayout.DEFAULT_SIZE, 699, Short.MAX_VALUE))
                .addContainerGap())
        );
        layout.setVerticalGroup(
            layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(layout.createSequentialGroup()
                .addContainerGap()

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
                    .addComponent(jLabel1)
                    .addComponent(cmbAdapters,     javax.swing.GroupLayout.PREFERRED_SIZE,
20, javax.swing.GroupLayout.PREFERRED_SIZE)
                    .addComponent(cmdStart2))
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                    .addComponent(jLabel2)
                    .addComponent(lblDeviceName,
javax.swing.GroupLayout.PREFERRED_SIZE,                                   23,
javax.swing.GroupLayout.PREFERRED_SIZE))
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                .addComponent(jTabbedPane1,       javax.swing.GroupLayout.PREFERRED_SIZE,
284, javax.swing.GroupLayout.PREFERRED_SIZE)
                .addGap(2, 2, 2)
                .addComponent(lstPackets,     javax.swing.GroupLayout.DEFAULT_SIZE,     169,
Short.MAX_VALUE)
                .addContainerGap())
        );
        pack();
    }// </editor-fold>//GEN-END:initComponents

    private void cmbAdaptersItemStateChanged(java.awt.event.ItemEvent evt) {//GEN-
FIRST:event_cmbAdaptersItemStateChanged
        // TODO add your handling code here:
        lblDeviceName.setText(devices[cmbAdapters.getSelectedIndex()].name);
    }//GEN-LAST:event_cmbAdaptersItemStateChanged

    private void cmdStart2ActionPerformed(java.awt.event.ActionEvent evt) {//GEN-
FIRST:event_cmdStart2ActionPerformed
// TODO add your handling code here:
        if(cmdStart2.getText().equals("Start")){
            pr=new packetReader();
            pr.start();
            cmdStart2.setText("Stop");
            cmbAdapters.setEnabled(false);
        }
        else{
            pr.stop();
            cmdStart2.setText("Start");
            cmbAdapters.setEnabled(true);
        }       // TODO add your handling code here:
}//GEN-LAST:event_cmdStart2ActionPerformed

    private void tblTCPMouseClicked(java.awt.event.MouseEvent evt) {//GEN-
FIRST:event_tblTCPMouseClicked
        // TODO add your handling code here:
        showTCPInfo();
    }//GEN-LAST:event_tblTCPMouseClicked

    void showTCPInfo(){
        if(tblTCP.getSelectedRow()!=-1){
            TCPPacket tcp=(TCPPacket)tcpPackets.get(tblTCP.getSelectedRow());
            lstPackets.removeAll();
            lstPackets.add("Source Host Name: " + tcp.src_ip.getHostName());
            lstPackets.add("Destination Host Name: " + tcp.dst_ip.getHostName());
            lstPackets.add("ACK: " + tcp.ack);
            lstPackets.add("ACK No.: " + tcp.ack_num);
            lstPackets.add("Dest. Port: " + tcp.dst_port);
            lstPackets.add("FIN flag: " + tcp.fin);
            lstPackets.add("TCP Option: " + tcp.option);
            lstPackets.add("PSH flag: " + tcp.psh);
            lstPackets.add("RST flag: " + tcp.rst);
            lstPackets.add("RSV flag1: " + tcp.rsv1);
            lstPackets.add("RSV2 flag: " + tcp.rsv2);
            lstPackets.add("Sequence: " + tcp.sequence);
            lstPackets.add("Src. Port: " + tcp.src_port);
            lstPackets.add("SYN flag: " + tcp.syn);
            lstPackets.add("URG flag: " + tcp.urg);
            lstPackets.add("Urgent Pointer: " + tcp.urgent_pointer);
            lstPackets.add("Window Size: " + tcp.window);
            lstPackets.add("Data: " + tcp.data);

        }
    }
    private void tblTCPKeyPressed(java.awt.event.KeyEvent evt) {//GEN-
FIRST:event_tblTCPKeyPressed
        // TODO add your handling code here:
    }//GEN-LAST:event_tblTCPKeyPressed

    private void cmdAddActionPerformed(java.awt.event.ActionEvent evt) {//GEN-
FIRST:event_cmdAddActionPerformed
        // TODO add your handling code here:
        if(!txtHostName.getText().equals("")){
            lstHosts.add(txtHostName.getText());
            txtHostName.setText("");
            txtHostName.requestFocus();
        }
    }//GEN-LAST:event_cmdAddActionPerformed

    private void lstBreachActionPerformed(java.awt.event.ActionEvent evt) {//GEN-
FIRST:event_lstBreachActionPerformed
        // TODO add your handling code here:
    }//GEN-LAST:event_lstBreachActionPerformed

    private void txtHostNameActionPerformed(java.awt.event.ActionEvent evt) {//GEN-
FIRST:event_txtHostNameActionPerformed
        // TODO add your handling code here:
    }//GEN-LAST:event_txtHostNameActionPerformed

    private void cmbAdaptersActionPerformed(java.awt.event.ActionEvent evt) {//GEN-
FIRST:event_cmbAdaptersActionPerformed
        // TODO add your handling code here:
    }//GEN-LAST:event_cmbAdaptersActionPerformed

    /**
     * @param args the command line arguments
     */
    public static void main(String args[]) {
        java.awt.EventQueue.invokeLater(new Runnable() {
            public void run() {
                new frmMain().setVisible(true);
            }
        });
    }

    //the packet capturing thread
    class packetReader extends Thread{
        @Override
        public void run(){
            System.out.println("n: " + devices.length);
        try{
          JpcapCaptor
captor=JpcapCaptor.openDevice(devices[cmbAdapters.getSelectedIndex()],    65535,    false,
20);
          while(true){
              Packet p=captor.getPacket();
              if(p!=null){
                  //getting the data link layer information
                  EthernetPacket etp=(EthernetPacket)p.datalink;
                  //implementation for TCP Packets
                  if(p.toString().indexOf("TCP")!=-1){
                      TCPPacket tcp=(TCPPacket)p;
                      if(tcp!=null){
                        tcpPackets.add(tcp);
                        DefaultTableModel model=(DefaultTableModel)tblTCP.getModel();
                        model.addRow(new Object[]{"","","",""});
                        //adding the source ip address
                        tblTCP.setValueAt(tcp.src_ip.getHostAddress(),tblTCP.getRowCount()-
1, 0);
                        //adding the source host name
                        tblTCP.setValueAt(etp.getSourceAddress(),tblTCP.getRowCount()-1, 1);
                         //adding the source ip address
                        tblTCP.setValueAt(tcp.dst_ip.getHostAddress(),tblTCP.getRowCount()-
1, 2);
                        //adding the source host name
                        tblTCP.setValueAt(etp.getDestinationAddress(),tblTCP.getRowCount()-
1, 3);
                        //getting the protocol name
                        if(tcp.src_port==20     ||     tcp.dst_port==20     ||     tcp.src_port==21     ||
tcp.dst_port==21){
                            tblTCP.setValueAt("FTP",tblTCP.getRowCount()-1, 4);
                        }
                        else if(tcp.src_port==80 || tcp.dst_port==80){
                            tblTCP.setValueAt("HTTP",tblTCP.getRowCount()-1, 4);
                        }
                        else if(tcp.src_port==110 || tcp.dst_port==110){
                            tblTCP.setValueAt("POP3",tblTCP.getRowCount()-1, 4);
                        }
                        else if(tcp.src_port==25 || tcp.dst_port==25){
                            tblTCP.setValueAt("SMTP",tblTCP.getRowCount()-1, 4);
                        }
                        else if(tcp.src_port==22 || tcp.dst_port==22){
                            tblTCP.setValueAt("SSH",tblTCP.getRowCount()-1, 4);
                        }
                        else if(tcp.src_port==23 || tcp.dst_port==23){
                            tblTCP.setValueAt("TELNET",tblTCP.getRowCount()-1, 4);
                        }
                      }
                      //checking breach of flag
                      for(int i=0;i<lstHosts.getItemCount();i++){
                          if(lstHosts.getItem(i).equals(tcp.dst_ip.getHostName())                    ||
lstHosts.getItem(i).equals(tcp.src_ip.getHostName())){
                              lstBreach.add("[" + new Date().toString() + "] Packet to/from flagged
host " + lstHosts.getItem(i));
                          }
                      }
                  }
              }
          }
      }
      catch(IOException ex){
          System.out.println(ex.toString());
      }

  }


}
```

```
    //class PacketPrinter implements PacketReceiver{
        //public void ReceivePacket(Packet packet){


        //}
    //}
    // Variables declaration - do not modify//GEN-BEGIN:variables
    private javax.swing.JComboBox cmbAdapters;
    private javax.swing.JButton cmdAdd;
    private javax.swing.JToggleButton cmdStart2;
    private javax.swing.JLabel jLabel1;
    private javax.swing.JLabel jLabel2;
    private javax.swing.JLabel jLabel3;
    private javax.swing.JPanel jPanel1;
    private javax.swing.JPanel jPanel2;
    private javax.swing.JPanel jPanel3;
    private javax.swing.JScrollPane jScrollPane1;
    private javax.swing.JTabbedPane jTabbedPane1;
    private javax.swing.JLabel lblDeviceName;
    private java.awt.List lstBreach;
    private java.awt.List lstHosts;
    private java.awt.List lstPackets;
    private javax.swing.JTable tblTCP;
    private javax.swing.JTextField txtHostName;
    // End of variables declaration//GEN-END:variables

}
```

CONCLUSION

Network Firewall System: *"A Network Security Tool"* A better firewall for the network administrator to monitor the LAN continuously and be able to analyze the activity over the network and thereafter will detect the malicious IP and block that IP in order to prevent the network from an intruder attack for the secure working of the network.

The suggested firewall with network monitoring will improve the visibility of the network activity and also make it very easy for the administrator to locate the fault and intrusion over the network and suitably correct either of them simultaneously by blocking the IP address of the faulty host.

The Firewall model discussed can be implemented for use in internal networks of various organizations and public domains, in order to monitor the correct working of the network without intrusion from the faulty host.

Network Analyzer can be used to strengthen the security of our network. Its resource identification like protocol detection is helpful to prevent security vulnerability in a certain extent. As mentioned earlier Breach of flag can be an extremely valuable network investigation tool, since many security holes are dependent on Breach of flag. But Network Analyzer is not free from demerits. For example if the packet makes a large number of jumps or hops our tool will not give correct information. But this is very rare in the network.

BIBLIOGRAPHY

Books:

- Deitel and Deitel. *Java- How To Program.*
- Herbert Schildt. *Java 2-Complete Reference*
- Andrew G. Blank.*TCP/IP JumpStart- Internet Protocol Basics*
- Bruce Eckel. *Thinking in Java*
- Andrew S. Tanenbaum. *Computer Networks 4th Ed.*
- Holzner. *Java 2 Black book Edition 2.* Dreamtech Press, Paraglyph Press 2006
- Darren Bolding .*Network security***.** *filters and firewalls*, September 1995

Research Paper:

Changhyun Baek, Eulgyu Im, Eungki Park, Kyunghee Choi, Gihyun Jung.
Design and Implementation of Firewall Simulation
based on SSFNet. Blocking illegal TCP connections.

Web Pages:

- www.wikipedia.org
- www.winpcap.politeo.com
- www.google.com
- www.java.sun.com
- www.javabeginner.com
- netresearch.ics.uci.edu/kfujii/jpcap/doc/tutorial