

FACULTY OF COMPUTER SCIENCE AND INFORMATION

TECHNOLOGY

UNIVERSITY OF MALAYA

WXES3182

Thesis

Session 2004/2005

**NETWORK SECURITY TOOLS –
FIREWALL**

DING KHOON CHONG

WEK020043

Supervisor

MR. LIEW CHEE SUN

Moderator

MR. ANG TAN FONG

ABSTRACT

A firewall is a system or group of systems that enforces an access control policy between two or more networks. The actual means by which this is accomplished varies widely, but in principle, the firewall can be thought of as a pair of mechanisms: one which exists to block traffic, and the other which exists to permit traffic. Some firewalls place a greater emphasis on blocking traffic, while others emphasize permitting traffic. Probably the most important thing to recognize about a firewall is that it implements an access control policy. If we do not have a good idea of what kind of access you want to allow or to deny, a firewall really would not help you. It is also important to recognize that the firewall's configuration, because it is a mechanism for enforcing policy, imposes its policy on everything behind it. Administrators for firewalls managing the connectivity for a large number of hosts therefore have a heavy responsibility.

ACKNOWLEDGEMENT

One of greatest pleasures of writing this report is acknowledging the efforts of many people whose hard work, cooperation, friendship, and understanding were crucial throughout the undergoing of this project.

Firstly, I would like to thank Mr. Liew Chee Sun, my supervisor for his guidance and advice throughout the entire project. The time that he has shared with me has made a great contribution to the success of the project. Following that, I would like to thank my considerate and kind moderator, Mr. Ang Tan Fong for his valuable pointers and comment on this project.

In addition, I would like to express my gratitude to my discussion group members. They are Mr.Kok Soo Leong, Mr.Gan Guan Sui, Mr.Gan Soon Huat for their willingness to share ideas and information during discussions session to make this project successful.

Finally yet importantly, I would like to express my appreciation to my course mates and family members for their encouragement and patience in the succession of this project.

TABLE OF CONTENTS

CHAPTER 1 INTRODUCTION

1.1 Project Overview.....	1
1.2 Project Motivation.....	1
1.3 Project Objective.....	2
1.4 Project Scope.....	3
1.5 Project Schedule.....	4
1.6 Report Layout.....	4

CHAPTER 2 LITERATURE REVIEW

2.1 Introduction of Networking.....	6
2.2 Network Structure.....	6
2.2.1 LAN.....	7
2.2.2 WAN.....	7
2.3 Computer Network Security Threat.....	8
2.3.1 Types of Attack.....	8
2.3.2 Attempts to Gain Access.....	9
2.3.4 Ping of Death.....	10
2.3.5 Viruses	10
2.3.6 Worm.....	11
2.3.7 Trojan.....	11
2.3.8 Spyware and Adware.....	14
2.4 TCP/IP.....	15

3.4 Analysis Workflow.....	60
3.5 Design Workflow.....	60
2.6 Implementation Workflow.....	61
2.7 Testing Workflow.....	61

CHAPTER 4 SYSTEM REQUIREMENTS ANALYSIS

4.1 Functional Requirements.....	63
4.1.1 Block Ping.....	63
4.1.2 Packet Filtering.....	63
4.1.3 Port Scanner.....	63
4.1.4 Generate Log file.....	64
4.1.5 GUI.....	64
4.1.6 Help Menu.....	64
4.1.7 System Tray.....	64
4.2 Non-Functional Requirement.....	65
4.3 Software and Hardware Selection.....	66
4.4 Development Requirements.....	67

CHAPTER 5 SYSTEM DESIGN

5.1 Deployment Diagram.....	68
5.2 FirePower Design Overview.....	69
5.3 FirePower Workflow.....	70
5.4 Context Diagram.....	71
5.5 FirePower Data Flow Diagram.....	72
5.6 FirePower User Interface Design.....	73

2.5	UDP.....	16
2.6	Network Protocol.....	17
2.6.1	FTP.....	17
2.6.2	IP.....	18
2.7	Socket.....	19
2.8	Computer Languages.....	20
2.8.1	C.....	22
2.8.2	C++.....	24
2.8.3	Java.....	26
2.8.4	C#.....	26
2.8.5	Network Address Translation (NAT).....	28
2.9	Case Study: Commercial Internet Firewall Product.....	28
2.9.1	Sygate.....	29
2.9.2	Tiny Firewall 6.0.....	31
2.9.3	ZoneAlarm.....	32
2.10	Firewall Technology.....	33
2.10.1	Packet Filtering.....	33
2.10.2	Circuit Level Gateways.....	40
2.10.3	Application Layer Firewalls.....	45
 CHAPTER 3 METHODOLOGY		
3.1	Overview.....	56
3.2	Process Model.....	57
3.2.1	Iterative-and-Incremental Model.....	57
3.3	Requirements Workflow.....	59

CHAPTER 6 SYSTEM IMPLEMENTATION

6.1 Introduction..... 76

6.2 Development Environment..... 76

6.3 Algorithms..... 77

6.4 Coding..... 77

6.4.1 The Filter-Hook Driver..... 77

6.4.2 Create the Kernel Mode Driver..... 78

6.4.2 Registering a Filter function..... 84

6.4.3 The Filter function..... 88

6.5 The reason why use this method to develop a Firewall..... 92

CHAPTER 7 SYSTEM TESTING

7.1 Introduction..... 93

7.2 Unit Testing..... 94

7.2.1 Module Testing..... 94

7.2.2 Integration Testing..... 96

7.2.2.1 Bottom - up Integration..... 97

7.3 System Testing..... 98

7.4 Conclusion..... 99

CHAPTER 8 SYSTEM EVALUATION

8.1 Introduction..... 100

8.2 System Strengths..... 100

8.2.1 Easy-to-use Application..... 100

8.2.2 Organized and User Friendly Interface..... 101

8.2.3 System Transparency..... 101

8.2.4 Reliable System with Effective Errors Handling..... 101

8.3 Problems Encountered and solutions..... 102

8.3.1 Difficulties in determining the scope of the application..... 102

8.3.2 Understanding on Current System Procedure..... 102

8.3.3 Difficulties in choosing a development platform..... 103

8.3.4 Lack of knowledge in develops Firewall..... 103

8.4 System Limitations..... 104

8.4.1 Lack of functional modules..... 104

8.4.2 Lack of Network Utilities Display..... 104

8.5 Future Enhancements..... 104

8.5.1 Increase the number of rules..... 105

8.5.2 Module Enhancement..... 105

8.6 Knowledge and experienced gained..... 105

8.7 Conclusion..... 106

REFERENCES 107

APPENDIXES 109

LIST OF FIGURES

Figure 2.1: TCP/IP Protocol Architecture.....	15
Figure 2.2: UDP in the OSI Reference Model.....	16
Figure 2.3: Six phases of C++ programming language.....	25
Figure 2.4: Sygate Personal Firewall Print Screen.....	29
Figure 2.5: Tiny Firewall 6.0 Print Screen.....	31
Figure 2.6: ZoneAlarm Print Screen.....	32
Figure 2.7: depicts the network packet evaluation process used by a packet filter firewall.....	36
Figure 2.8: depicts the network packet evaluation process used by a circuit level firewall.....	41
Figure 2.9: depicts the network packet evaluation process used by a application layer firewall.....	46
Figure 2.10: depicts the flow of communications between a real client and a network server when the communications pass through a proxy service.....	48
Figure 2.11: depicts the network packet evaluation process used by a dynamic packet filter firewall.....	54
Figure 3.1: Iterative-and-Incremental Model.....	59
Figure 4.1: Components of Net Defender.....	62
Figure 5.1: an Overview of the FirePower is Placed.....	68
Figure 5.2: FirePower Architecture.....	69
Figure 5.3: Showing FirePower Workflow.....	70

Figure 5.4: FirePower Context Diagram..... 71

Figure 5.5: FirePower Data Flow Diagram 72

Figure 5.6: Initial FirePower Interface..... 74

Figure 7.1 System Testing Steps..... 93

Figure 7.2 Process Monitoring Module..... 95

Figure 7.3 Simple Port Scanner Module Integrating Test..... 96

Figure 7.4 Bottom-up Testing..... 97

Figure 7.5 Insert Rule to FirePower..... 98

Figure 7.6 ICMP Packet is blocked..... 99

CHAPTER 1

INTRODUCTION

1.1 Project Overview

This chapter describes the purposes of the project and the problems to be solved. The system functions and rationale of the project will be discussed too later in this chapter.

The aim of this project is to define a firewall. The firewall in an organization uses provides the security at the entranceway to network. If you permit individuals to use modems, your organization does not have a single entryway, but many ports of exposure. The firewall should function at the level required by your policy for authenticating traffic, collecting sufficiently detailed logging, and perhaps inspection of data which passes (for viruses or applets).

Most of all, a firewall is an implementation of security policy. The policy itself should be based on examining your organization's assets, and determine which level of access is appropriate when compared to the risk to those assets.

1.2 Project Motivation

In order to guard the traffic of whether it is from outside or inside of a large computer network, a firewall is necessary to prevent spyware or unauthorized intruders to access a certain computer system. A computer network firewall is an electronic blocking mechanism that will not allow unauthorized intruders into a computer system. A computer firewall is a software program that blocks potential hackers from your individual computer or your computer network. Many different computer firewall software packages are

available with a broad variety of costs and update options. Any computer that is always connected to the internet needs a firewall package.

1.2 Project Scope

Firewall Software is a basic requirement for anyone using broadband to prevent hacking, virus, and other security risks. Firewall software is software designed to prevent unauthorized access to a computer or network that is connected to the Internet. Firewall software comes in a variety of forms, offering a wide variety of features, protection capabilities, scalability and cost. From personal firewall software that can be purchased for a few dollar, to corporate firewall software costing thousands, firewall software is a necessary component of having any type of broadband connection. Typically, firewall software works by hiding your computer (via the ports that connect it to the Internet) from unknown users. Firewall software basically 'stealths' your computer or network, hiding it from hackers who scour the Internet looking for vulnerable computers that they can gain access to.

1.3 Project Objective

- Do research on firewall technologies.
- Develop a firewall filtering uses a range of information in the packet header (for example, source and destination IP addresses, port, and protocol) while routing uses only the destination IP address.
- Detects and Blocks malicious Internet activity before it can reach computer.
- Identifies attackers and type of attack by IP.
- Protects personal information from hackers.
- Alerts and Records threatening Internet traffic to computer.

1.5 Project Schedule

ID	Task Name	Start	Finish	Duration	2004						2005			
					Jul	Aug	Sep	Oct	Nov	Dec	Jan	Feb	Mar	Apr
1	Project Definitions	6/16/2004	7/27/2004	30d										
2	Literature Review	6/25/2004	7/29/2004	25d										
3	Research	7/19/2004	8/27/2004	30d										
4	System Analysis	8/18/2004	9/9/2004	17d										
5	System Design	9/1/2004	10/1/2004	23d										
6	Implementation	10/1/2004	1/6/2005	70d										
7	Testing	1/2/2004	2/5/2004	25d										
8	Documentation	7/26/2004	2/18/2005	150d										

1.6 Report Layout

This report has been divided to 5 chapters, which are organized as follows:

i. Chapter 1 Introduction

Introduces the current networking technologies associates with this project and FirePower, project definition, project objectives, project schedule and report layout.

ii. Chapter 2 Literature Review

This will include all the related topic to firewall and also case study of the commercial internet firewall product as a reference.

iii. Chapter 3 Methodology

Define the Methodology used for FirePower development. A Methodology will offer a step-by-step approach to the desired situation – and each step will offer some success and benefit.

iv. Chapter 4 System Requirements Analysis

Explain what to build firewalls using different technologies that are available. It also presents the evaluation criteria should apply when developing the most appropriate firewall.

v. Chapter 5 System Design

Overview of Internet firewall design in order to produce firewall solution for our machine.

vi. Chapter 6 System Implementation

Explain fundamentally how filtering driver works.

vii. Chapter 7 System Testing

Overview of implementing system testing throughout the whole development.

viii. Chapter 8 System Evaluation

Define the strengths and weaknesses of the system and follow by indicating future enhancement.

CHAPTER 2 LITERATURE REVIEW

2.1 Introduction of Networking

In the world of computers today, networking is the practice of linking two or more computing devices together for the purpose of sharing data. Networks are built with a mix of computer hardware and computer software.

Computer networks come in many different shapes and sizes. Over the years, the networking industry has coined terms like "LAN" and "WAN" attempting to define sensible categories for the major types of network designs. The precise meaning of this terminology remains lost on the average person, however.

2.2 Network Structure

The industry refers to nearly every type of network as an "area network." The most commonly-discussed categories of computer networks include the following:

- i. Local Area Network (LAN)
- ii. Wide Area Network (WAN)

2.2.1 LAN

A LAN connects network devices over a relatively short distance. A networked office building, school, or home usually contains a single LAN, though sometimes one building will contain a few small LANs, and occasionally a LAN will span a group of nearby buildings. In IP networking, one can conceive of a LAN as a single IP subnet.

Besides operating in a limited space, LANs include several other distinctive features. LANs are typically owned, controlled, and managed by a single person or organization. They also use certain specific connectivity technologies, primarily Ethernet and Token Ring.

2.2.2 WAN

As the term implies, a wide-area network spans a large physical distance. A WAN like the Internet spans most of the world!

A WAN is a geographically-dispersed collection of LANs. A network device called a router connects LANs to a WAN. In IP networking, the router maintains both a LAN address and a WAN address.

2.3 Computer Network Security Threat

2.3.1 Types of Attack

Before determining exactly what type of firewall we need, we must first understand the nature of security threats that exist. The Internet is one large community, and as in any community it has both good and bad elements. The bad elements range from incompetent outsiders who do damage unintentionally, to the proficient, malicious hackers who mount deliberate assaults on companies using the Internet as their weapon of choice.

Generally there are three types of attack that could potentially affect our business:

- Information theft: Stealing company confidential information, such as employee records, customer records, or company intellectual property
- Information sabotage: Changing information in an attempt to damage an individual or company's reputation, such as changing employee medical or educational records or uploading derogatory content onto your Web site
- Denial of service (DoS): Bringing down your company's network or servers so that legitimate users cannot access services or so that normal company operations such as production are impeded

2.3.2 Attempts to Gain Access

A hacker may attempt to gain access for sport or greed. An attempt to gain access usually starts with gathering information about the network. Later attacks use that information to achieve the real purpose—to steal or destroy data.

A hacker may use a port scanner—a piece of software that can map a network. It is then possible to find out how the network is structured and what software is running on it.

Once the hacker has a picture of the network, he can exploit known software weaknesses and use hacking tools to wreak havoc. It is even possible to get into the administrator's files and wipe the drives, although a good password will usually foil that effort.

Fortunately, a good firewall is immune to port scanning. As new port scanners are developed to get around this immunity, firewall vendors produce patches to maintain the immunity.

2.3.3 Denial-of-Service Attacks

DoS attacks are purely malicious. They don't result in any gain for the hacker other than the "joy" of rendering the network, or parts of it, unavailable for legitimate use. DoS attacks overload a system so that it isn't available—they deny your ability to use your network service. To overload the system, the hacker sends very large packets of data or programs that require the system to respond continuously to a bogus command.

To launch a DoS attack, a hacker must know the IP address of the target machine. A good firewall doesn't reveal its own IP address or the IP addresses on the LAN. The hacker may think he has contacted the network when he has only contacted the firewall—and he can't lock up the network from there. Furthermore, when a hacker launches an attack, some firewalls can identify the incoming data as an attack, reject the data, alert the system administrator, and track the data back to the sender, who can then be apprehended.

Advanced antivirus software programs exist to combat viruses. Antivirus software examines the contents of local hard drives to identify patterns of data called "signatures" that match known viruses.

2.3.6 Worm

Worms are malicious software applications designed to spread via computer networks. Worms are one form of malware along with viruses and Trojans. A person typically installs worms by inadvertently opening an email attachment or message that contains executable scripts.

Once installed on a computer, worms spontaneously generate additional email messages containing copies of the worm. They may also open TCP ports to create networks security holes for other applications, and they may attempt to "flood" the LAN with spurious Denial of Service (DoS) data transmissions.

Being embedded inside everyday network software, worms easily penetrate most firewalls and other network security measures.

2.3.7 Trojan

Named after the Trojan Horse of ancient Greek history, a Trojan is a network software application designed to remain hidden on an installed computer. Trojans generally serve malicious purposes and are therefore a form of malware.

Trojans sometimes, for example, access personal information stored locally on home or business computers, then send these data to a remote party via the Internet. Alternatively, trojans may serve merely as a "backdoor" application, opening network ports to allow other network applications access to that computer. Trojans are also capable of launching

2.3.4 Ping of Death

In late 1996 and early 1997, a flaw in the implementation of networking in some operating systems became well-known and popularized by hackers as a way to crash computers remotely over the Internet. The Ping of Death attack was relatively easy to carry out and very dangerous due to its high probability of success.

Technically speaking, the Ping of Death attack involved sending IP packets of a size greater than 65,535 bytes to the target computer. IP packets of this size are illegal, but applications can be built that are capable of creating them. Carefully programmed operating systems could detect and safely handle illegal IP packets, but some failed to do this. ICMP ping utilities often included large-packet capability and became the namesake of the problem, although UDP and other IP-based protocols also could transport Ping of Death.

2.3.5 Viruses

In computer technology, viruses are malicious software programs, a form of malware. Viruses exist on local disk drives and spread from one computer to another through sharing of "infected" files. Common methods for spreading viruses include floppy disks, FTP file transfers, and copying files between shared network drives.

Once installed on a computer, a virus may modify or remove application and system files. Some viruses render a computer inoperable; others merely display startling screen messages to unsuspecting users.

Denial of Service (DoS) attacks. A combination of firewalls and antivirus software protect networks against trojans.

In contrast to worms, however, trojans do not replicate themselves or seek to infect other systems once installed on a computer.

How Trojans Work

Today's security challenges are mainly related to preventing Trojans and Worms from infiltrating the computer and causing various damages on it including the theft of data. There are several stages for hackers to exploit your computer.

1. Get to the target computer and inject the malicious code

This is the most difficult part for the hacker. There are few examples of the exploits (with the comments on protection):

- RPC service buffer overflow (MSBlast) => sending special packets to port 135 forces RPC service to execute some payload (= code) of the packets.
- Outlook Express & Outlook MIME exploit => sending special html email forces automatically execute attached exe.
- Could use other exploits of MS applications bugs (IIS, IE, OE....)
- VBA macro in MS office documents (requires user to open the document)
- email attachment (exe, vbs, js... - requires user to open the attachment).
- web page active content (requires user to visit the web page and sometimes also to confirm the active code download).

- modifying content of known binary file such as winword.exe (this is classical old fashioned virus) e.g. on a common network share.
- boot sectors of floppy disks or other removable media is old fashioned yet successful approach (the code automatically executes when the drive is mounted).

2. Let the injected code "install" the Trojan itself

Usually the injected code has only one opportunity to execute (email won't be opened again; user will not run the .exe or visit the particular web page again etc.). Therefore the code (worm) must establish a way of how to start again. The worm copies itself into some location, usually within Windows subtree (c:\windows etc.), and creates some unsuspecting looking name for itself (e.g. Microsoft known program names such as iis.exe or dllhost.exe - the latter one is in different dir to avoid problems in overwriting the original file).

To maintain the future start the worm executes some of the following scenarios:

- Use Run keys to autostart on next login.
- Installs as a service or (rarely) as a driver (either via API or directly writing into registry).
- Using other keys to autostart in some occasions (e.g. exefile).
- copies itself into Start Menu\Programs\Startup folder
- Replaces .jpg (or other) files with xxxx.jpg.exe (the last extension is hidden in explorer by default) thus waiting till user opens some picture.
- One JScript virus has used ActiveX control with the CLSID "06290BD5-48AA-11D2-8432-006008C3FBFC" to get control over files.

2.3.8 Spyware and Adware

Spyware and Adware is software made by publishers that allow them to snoop on your browsing activity, invade your privacy, and flood you with those horrible popups. If you are like most users on the internet, chances are you are probably infected with these applications. That is why we have designed our revolutionary product.

Spyware and Adware affect every internet user by:

- All information you enter via the web can be intercepted.
- Unauthorized sites can add themselves to your desktop (icons).
- Unauthorized sites can add themselves to your internet favorites.
- Your browsing activity can be tracked and monitored.
- Unwanted toolbars and searchbars can attach themselves to your browser without your knowledge or approval.
- Your personal information can be sold to other parties without your knowledge or consent.
- Your default homepage and settings can be hijacked so you can't change them.
- These malicious components not only invade your PC so they can not be removed, but take up your hard drive space and slow down your PC.

2.4 TCP/IP

TCP/IP protocols map to a four-layer conceptual model known as the DARPA model. The four layers of the DARPA model are: Application, Transport, Internet, and Network Interface. Each layer in the DARPA model corresponds to one or more layers of the seven-layer Open Systems Interconnection (OSI) model.

Transmission Control Protocol (TCP) and Internet Protocol (IP) are two distinct network protocols, technically speaking. TCP and IP are so commonly used together, however, that TCP/IP has become standard terminology to refer to either or both of the protocols. In other words, the term TCP/IP refers to network communications where the TCP transport is used to deliver data across IP networks.

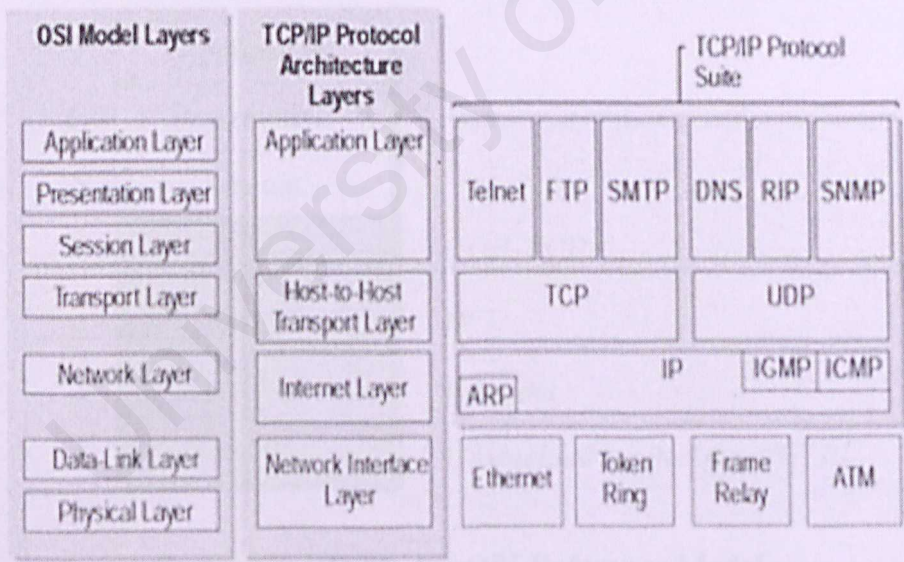


Figure 2.1:TCP/IP Protocol Architecture

2.5 UDP

The User Datagram Protocol (UDP) supports network applications that need to transport data between computers. Applications that use UDP include client/server programs like video conferencing systems. Although UDP has been in use for many years -- and overshadowed by more glamorous alternatives -- it remains an interesting and viable technology.

UDP -- like its cousin the Transmission Control Protocol (TCP) -- sits directly on top of the base Internet Protocol (IP). Recalling the Open Systems Interconnection (OSI) model of networking, UDP (and TCP) are transport layer protocols as shown below.

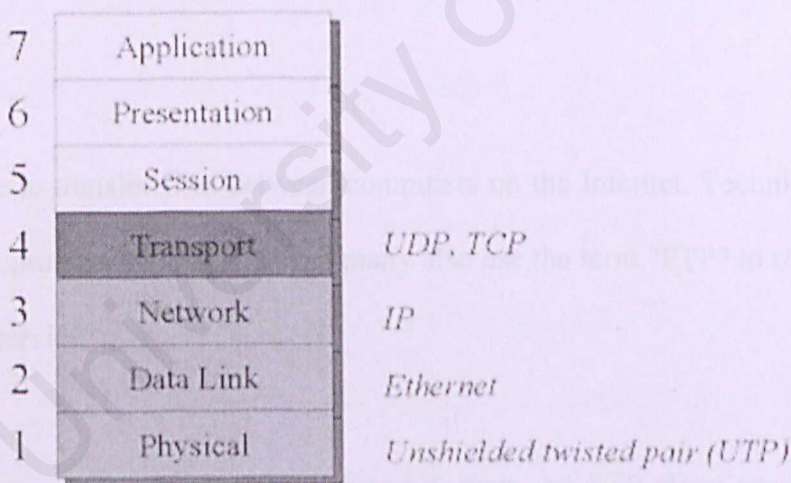


Figure2.2: UDP in the OSI Reference Model

In general, UDP implements a fairly "lightweight" layer above the Internet Protocol. UDP's main purpose is to abstract network traffic in the form of datagrams. A datagram comprises one single "unit" of binary data; the first eight (8) bytes of a datagram contain the header information and the remaining bytes contain the data itself.

2.6 Network Protocol

A network protocol defines a "language" of rules and conventions for communication between devices. A protocol includes formatting rules that specify how data is packaged into messages. It also may include conventions like message acknowledgement or data compression to support reliable and/or high-performance communication.

Many protocols exist in computer networking ranging from high level (like SOAP) to low level (like ARP). The Internet Protocol family includes IP and all higher-level network protocols built on top of it, such as TCP, UDP, HTTP, and FTP. Modern operating systems include services or daemons that implement support for specific protocols. Some protocols, like TCP/IP, have also been implemented in silicon hardware for optimized performance.

2.6.1 FTP

FTP allows one to transfer files between computers on the Internet. Technically, FTP is a simple network protocol based on IP, but many also use the term "FTP" to refer to this type of file sharing service.

The FTP service is based on client/server architecture. An FTP client program initiates a connection to a remote computer running FTP server software. After the connection is established, the client can choose to send and/or receive copies of files, singly or in groups. To connect to an FTP server, a client generally requires a username and password as set by the administrator of the server. Many public FTP archives follow a special convention for that accepts a username of "anonymous."

FTP clients are included with most network operating systems, but most operating system clients (such as FTP.EXE on Windows) support a relatively unfriendly command-line interface. Many freeware and shareware third-party FTP clients have been developed that support graphic user interfaces (GUIs) and additional convenience features. In either command-line or graphic interfaces, FTP clients identify the server either by its IP address (such as 192.168.0.1) or by its host name (such as ftp.about.com).

The FTP protocol supports two modes of data transfer: plain text (ASCII), and binary. The mode an FTP client uses must generally be configured by the end user. The mode usually defaults to plain text. The most common error one makes in using FTP occurs when attempting to transfer a binary file (such as a program or music file) while in text mode. A copy of the file is made, but this copy will often be unusable. When working with FTP clients and files, learn to use the transfer mode properly.

2.6.2 IP

IP is probably the world's single most popular network protocol. Data travels over an IP-based network in the form of packets; each IP packet includes both a header (that specifies source, destination, and other information about the data) and the message data itself.

IP supports the notion of unique addressing for computers on a network. Current IP (IPv4) addresses contain four bytes (32 bits) that is sufficient to address most computers on the Internet.

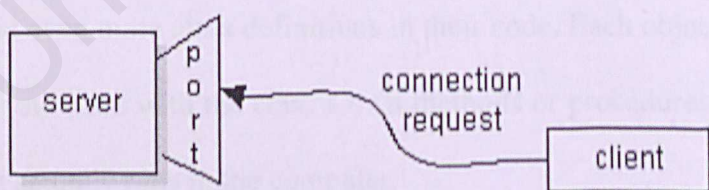
IP supports protocol layering as defined in the OSI reference model. Popular higher-level protocols like HTTP, TCP, and UDP are built directly on top of IP. Likewise, IP can travel over several different lower-level data link interfaces like Ethernet and ATM. IP originated with UNIX® networking in the 1970s.

2.7 Socket

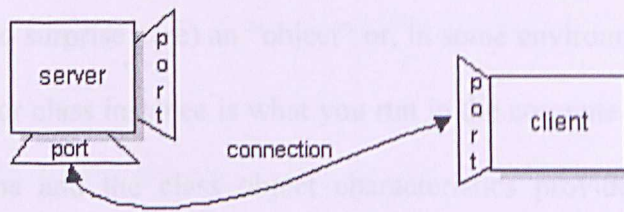
A socket is one end-point of a two-way communication link between two programs running on the network. Socket classes are used to represent the connection between a client program and a server program.

Normally, a server runs on a specific computer and has a socket that is bound to a specific port number. The server just waits, listening to the socket for a client to make a connection request.

On the client-side: The client knows the hostname of the machine on which the server is running and the port number to which the server is connected. To make a connection request, the client tries to rendezvous with the server on the server's machine and port.



If everything goes well, the server accepts the connection. Upon acceptance, the server gets a new socket bound to a different port. It needs a new socket (and consequently a different port number) so that it can continue to listen to the original socket for connection requests while tending to the needs of the connected client.



On the client side, if the connection is accepted, a socket is successfully created and the client can use the socket to communicate with the server. Note that the socket on the client side is not bound to the port number used to rendezvous with the server. Rather, the client is assigned a port number local to the machine on which the client is running.

2.8 Computer Languages

Object-oriented Programming

Object-oriented Programming (OOP) is organized around “objects” (a software bundle of variable and related methods) rather than “actions,” data rather than logic. Objects are the things you think about first in designing a program and they are also the units of code that are eventually derived from the process. In between, each object is made into a generic class of object and even more class definitions in their code. Each object is an instance of a particular class or subclass with the class’s own methods or procedures and data variables. An object is what actually runs in the computer.

The first step in OOP is to identify all the objects you want to manipulate and how they relate to each other, an exercise often known as data modeling. Once you’ve identified an object, you generalize it as a class of objects (think of Plato’s concepts of the “ideal” chair that stands for all chairs) and define the kind of data it contains and any logic sequences

that can manipulate it. Each distinct logic sequence is known as a method. A real instance of a class is called (no surprise here) an “object” or, in some environments, an “instance of a class.” The object or class instance is what you run in the computer. Its methods provide computer instructions and the class object characteristics provide relevant data. You communicate with objects- and they communicate with each other- with well-defined interfaces called messages.

The concepts and rules used in object-oriented programming provide these important benefits:

- i. The concept of a data class makes it possible to define subclasses of data objects that share some or all of the main class characteristics. Called inheritance, this property of OOP forces a more thorough data analysis, reduces development time, and ensures more accurate coding.
- ii. Since a class defines only the data it needs to be concerned with, when an instance of that class (an object) is run, the code will not be able to accidentally access other program data. This characteristic of data hiding provides greater system security and avoids unintended data corruption.
- iii. The definition of a class is reusable not only by the program for which it is initially created but also by other object-oriented programs (and, for this reason, can be more easily distributed for use in networks).
- iv. The concept of data classes allows a programmer to create any new data type that is not already defined in the language itself.
- v. One of the first object-oriented computer languages was called Smalltalk. C++ and Java are the most popular object-oriented languages today. The Java

programming languages is designed especially for use in distributed applications on corporate networks and the Internet.

The following describe some advantages of object-oriented programming language.

- i. Simplicity: software objects model real world objects, so the complexity is reduced and the program structure is very clear.
- ii. Modularity: each object forms a separate entity whose internal workings are decoupled from other parts of the system.
- iii. Modifiability: it is easy to make minor changes in the data representation or procedures in an OO program. Changes inside a class do not affect any other part of a program, since the only public interface that the external world has to a class is through the use of methods.
- iv. Extensibility: adding new features or responding to changing operating environment can be solved by introducing a few new objects and modifying some existing ones.
- v. Maintainability: objects can be maintained separately, making locating and fixing problems easier.
- vi. Reusability: objects can be reused in different programs.

2.8.1 C

The C programming language was devised in the early 1970s as a system implementation language for the nascent UNIX operating system. Derived from the typeless language BCPL, it evolved a type structure; created on a tiny machine as a tool to improve programming environment, it has become one of the dominant languages of today.

As a programming language, C is rather like Pascal or FORTRAN. Values are stored in variables. Programs are structured by defining and calling functions. Program flow is controlled using loops, if statements and function calls. Input and output can be directed to the terminal or to files. Related data can be stored together in arrays or structures.

Of the three languages, C allows the most precise control of input and output. C is also rather terser than FORTRAN or Pascal. This can result in short efficient programs, where the programmer has made wise use of C's range of powerful operators. It also allows the programmer to produce programs which are impossible to understand.

Programmers who are familiar with the use of pointers (or indirect addressing, to use the correct term) will welcome the ease of use compared with some other languages. Undisciplined use of pointers can lead to errors which are very hard to trace. This course only deals with the simplest applications of pointers.

C programs will look similar under any other system (such as VMS or DOS), some other features will differ from system to system. In particular the method of compiling a program to produce a file of runnable code will be different on each system.

The UNIX system is itself written in C. In fact C was invented specifically to implement UNIX. All of the UNIX commands which you type, plus the other system facilities such as password checking, lineprinter queues or magnetic tape controllers are written in C.

In the course of the development of UNIX, hundreds of functions were written to give access to various facets of the system. These functions are available to the programmer in libraries. By writing in C and using the UNIX system libraries, very powerful system programs can be created. These libraries are less easy to access using other programming languages. C is therefore the natural language for writing UNIX system programs.

2.8.2 C++

C++ is a general purpose programming language with a bias towards systems programming that

- is a better C
- supports data abstraction
- supports object-oriented programming
- supports generic programming.

C++ is an object-oriented programming (OOP) language that is viewed by many as the best language for creating large-scale applications. C++ is a superset of the C language.

The C++ language facilitates structured and disciplined approach to computer program design. C++ programs consist of pieces called classes and function. Programmer can program each piece that programmer need to form a C++ program. But most C++ programmers take advantage of the rich collections of existing classes and functions in the C++ standard library. C++ programs typically go through six phases to be executed. These are: edit, preprocess, compile, link, load, and execute. (Deitel, 1997)

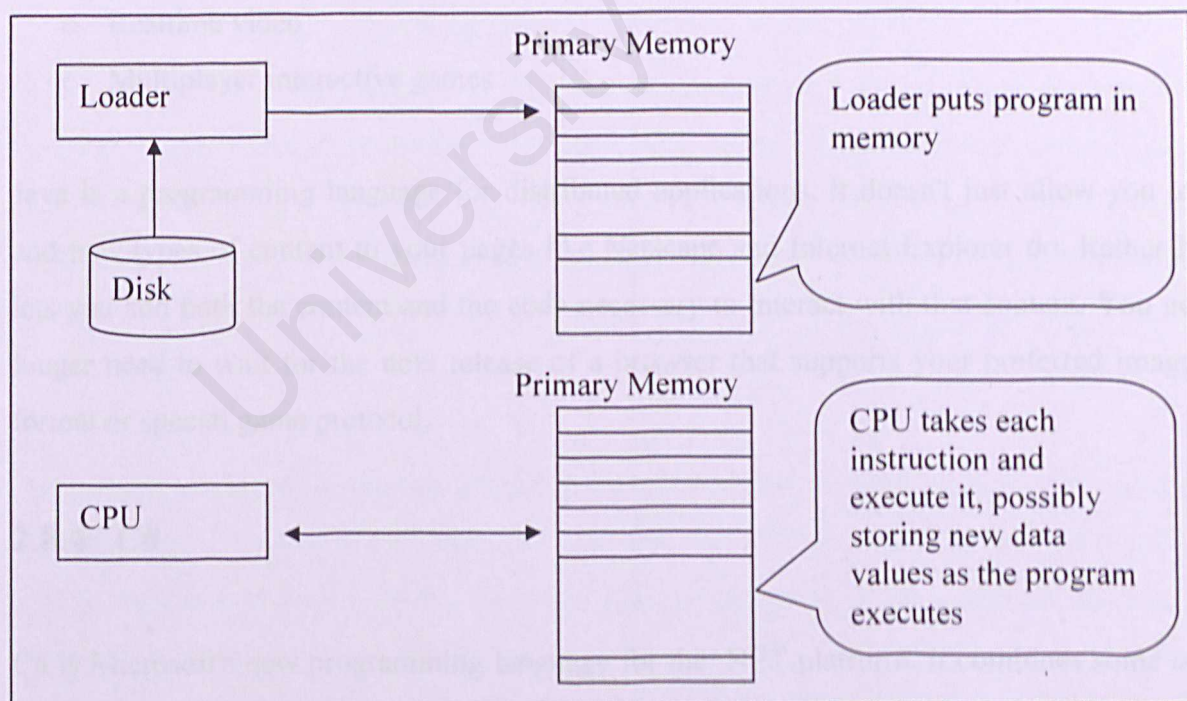
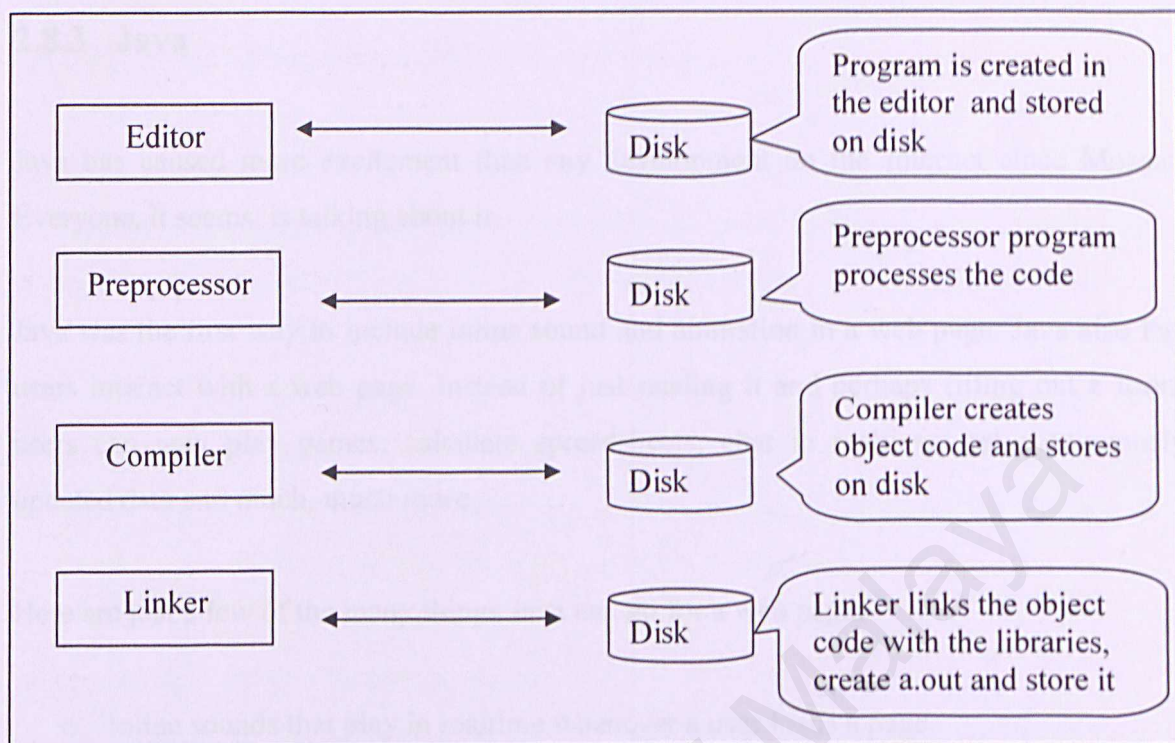


Figure 2.3: Six phases of C++ programming language

2.8.3 Java

Java has caused more excitement than any development on the Internet since Mosaic. Everyone, it seems, is talking about it.

Java was the first way to include inline sound and animation in a web page. Java also lets users interact with a web page. Instead of just reading it and perhaps filling out a form, users can now play games, calculate spreadsheets, chat in realtime, get continuously updated data and much, much more.

Here are just a few of the many things Java can do for a web page:

- Inline sounds that play in realtime whenever a user loads a page
- Music that plays in the background on a page
- Cartoon style animations
- Realtime video
- Multiplayer interactive games

Java is a programming language for distributed applications. It doesn't just allow you to add new types of content to your pages like Netscape and Internet Explorer do. Rather it lets you add both the content and the code necessary to interact with that content. You no longer need to wait for the next release of a browser that supports your preferred image format or special game protocol.

2.8.4 C#

C# is Microsoft's new programming language for the .NET platform. It combines some of the best features of modern programming languages such as Java, C++ or Visual Basic. C# is an object-oriented language with single inheritance but multiple interfaces per class. It supports component-based programming by properties (smart fields), events and delegates

(enhanced function pointers). C# is fully interoperable with other .NET languages such as VB.NET, Eiffel.NET or Oberon.NET.

Briefly, here are several features of .Net that make a suitable and robust environment for application development on supported platforms, currently Windows. We'll explore each of these points in detail in the next few lessons of this tutorial.

i. Cross Language Support.

All .Net languages are based on the same underlying type system. This means that code written in one .Net language can easily use and integrate with code written in other .Net languages. .Net languages include C#, Visual Basic.Net and managed C++.

ii. Use of Common Internet protocols.

.Net offers extensive support for XML, which is the choice for formatting information over the Internet. Additionally, support for transfer via SOAP is also integrated.

iii. Use of Metadata within Assemblies.

.Net components (Executables and libraries in C++ lingo) are deployed as part of assemblies. Each assembly contains metadata that allows simple assembly versioning, simpler reflections to determine assemble content and capabilities and component based security. These are real advantages over what Java offers and we will explore them in detail.

iv. Simple Deployment

The metadata in assemblies also simplifies deployment. An assembly can specify exactly the location and version of any other code it needs. The problems of maintaining a registry such as that needed with .COM components and the problems of DLL version mismatch have been eliminated.

v. Type Checking

The CLR, common language runtime, type checks all objects in use. All objects are derived from an object class, similar to what is done in Java.

Memory Management and Garbage Collection

The CLR also handles memory allocation and garbage collection. Garbage collection is the automatic reclaiming of memory from objects that are no longer in scope. The memory management nightmares of C++ are gone.

2.8.5 Network Address Translation (NAT)

Firewalls using NAT and/or Port Address Translation (PAT) completely hide the network protected by the firewall by using many-to-one address translation. In most NAT implementations there is a single public IP address used for the entire network. All packets going outside the network have their internal IP addresses hidden for security, so any incoming packets are delivered to the network's public IP address. To handle ensuing port conflicts, PAT needs to be added to NAT.

A disadvantage of NAT is that it can't properly pass protocols containing IP address information in the data portion of the packet.

2.9 Case Study: Commercial Internet Firewall Product

Today there are many commercial internet personal firewalls at the market. Each has its own strengths and weakness. So you simply choose one of them that meet your needs install into your computer, then doing some simple configuration finally your computer being protected from malicious intruder. The following are some examples of today commercial personal firewall.

2.9.1 Sygate

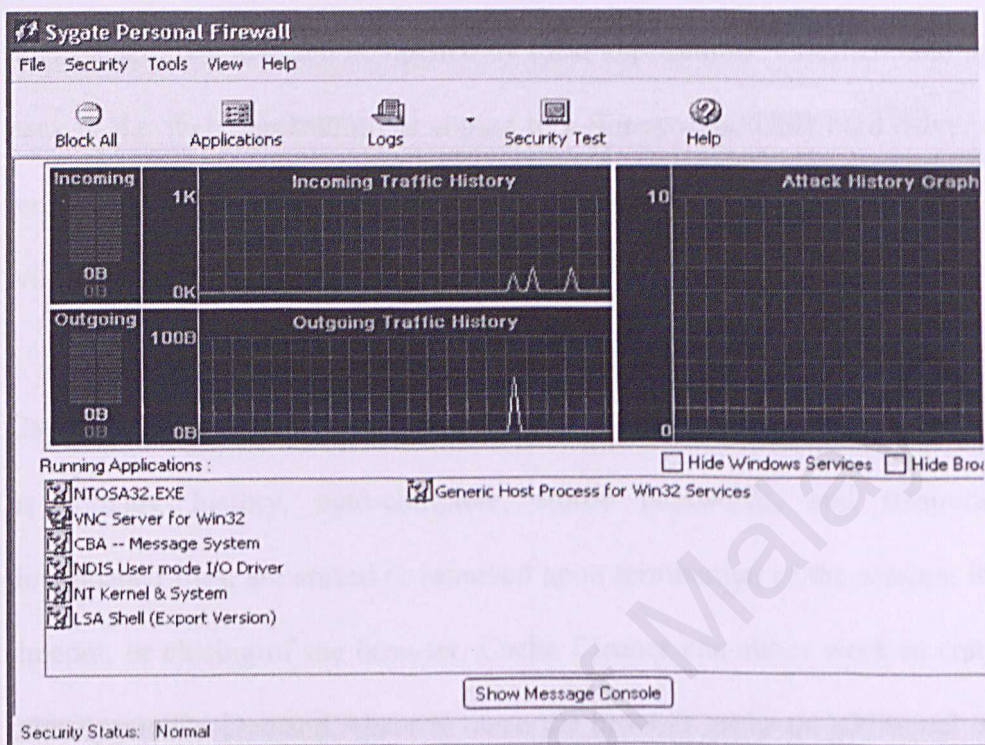


Figure 2.4: Sygate Personal Firewall Print Screen

The Sygate solution provides comprehensive protection against information leakage by controlling the network communication of all managed endpoints, preventing unauthorized P2P file transfer applications, analyzing the content of traffic for sensitive data, and ensuring that all sensitive information generated on or delivered via Web applications to 3rd-party-owned endpoints is encrypted on the endpoint, and sanitizes at the end of the Web session, preventing recovery of the deleted files. Thus, information leakage is avoided, and data privacy is preserved. There some extra features with Sygate:

- i. **Host Integrity:** Host Integrity ensures that devices accessing confidential data are secured by antivirus software with updated virus definitions, a personal firewall, critical service packs, and patches.

- ii. Virtual Desktop: The Virtual Desktop creates secure encrypted environment on the endpoint that enables users to download confidential data into a virtual environment where it can be opened by local applications, modified, and uploaded back to the Web application, or copied to a floppy disk, USB hard drive, or other removable media. When the session is terminated or times out, the virtual desktop will sanitize the system, removing all data generated during the session.
- iii. Cache Cleaner: Sygate Cache Cleaner ensures that Web browser information, such as cookies, history, auto-complete, stored passwords, and temporary and downloaded files, are erased or removed upon termination of the session, inactivity timeout, or closing of the browser. Cache Cleaner can either work in conjunction with Sygate On-Demand Agent to clean the browser cache on additional operating systems such as Mac OSX, Linux, and Windows (98,ME), or as a standalone module.

2.9.2 Tiny Firewall 6.0

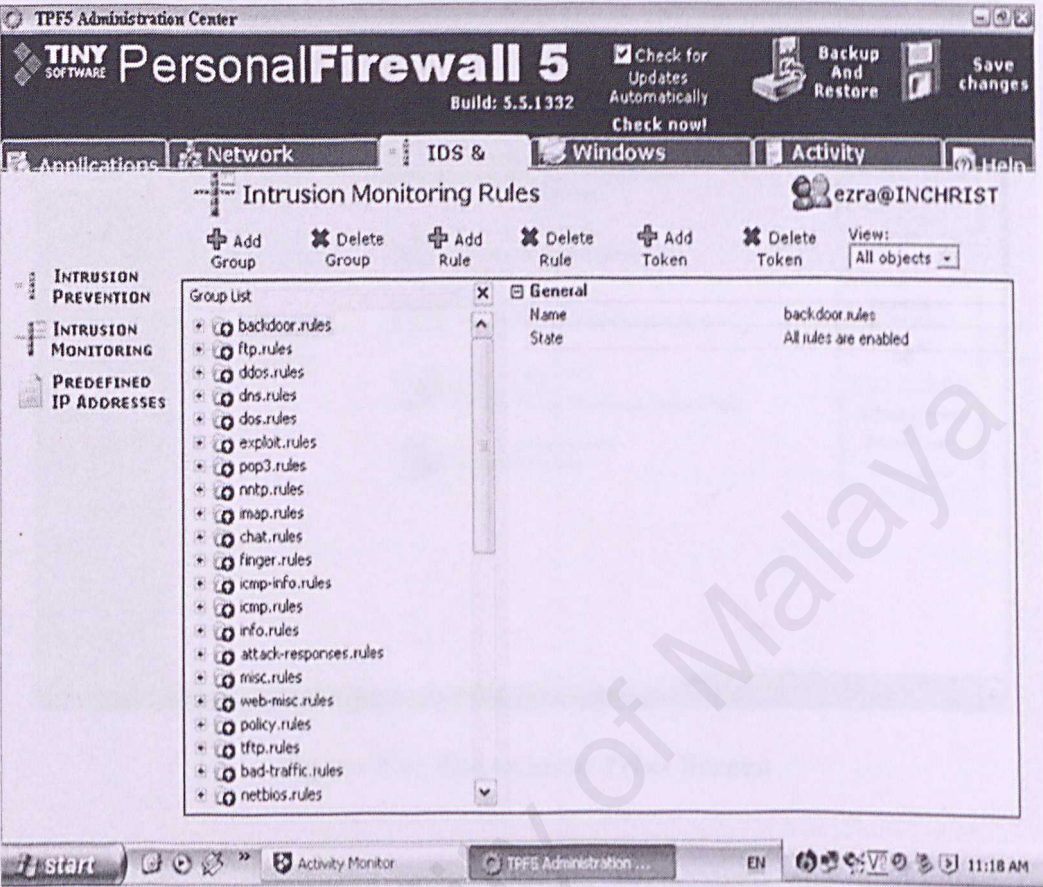


Figure 2.5: Tiny Firewall 6.0 Print Screen

Several functions for Tiny Firewall are:

- i. Blocks Network Activity
- ii. Stops Unknown Viruses and Spyware
- iii. Controls Application Behavior
- iv. Protects Files and Computer Settings
- v. Multiuser Environment

2.9.3 ZoneAlarm

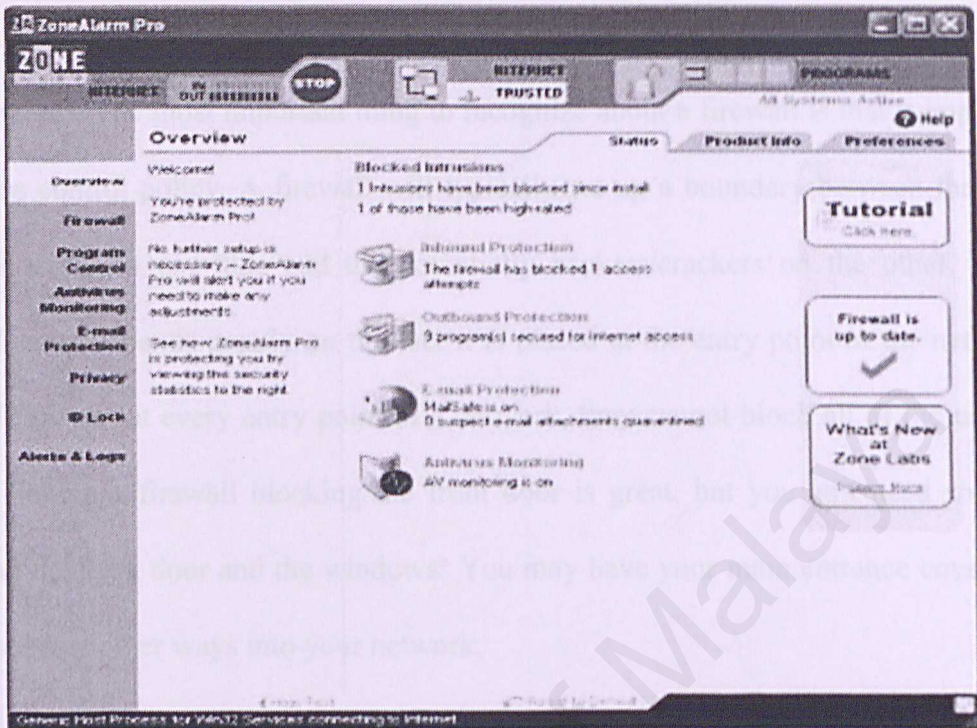


Figure 2.6: ZoneAlarm Print Screen

- i. Zone Labs is the award-winning leader in personal and distributed firewall protection for over 25 million PCs worldwide.
- ii. Zone Labs now protects you and your PC across all major Internet activities, including email, Web surfing, online shopping, banking, instant messaging, and more.
- iii. Zone Labs products are easy to use, yet powerful. Novices can protect themselves with little effort because Zone Labs products come with optimized, out-of-the-box security settings. Savvy users can fine-tune Internet security controls to meet their precise needs.

2.10 Firewall Technology

A firewall is a system or group of systems that enforces an access control policy between two networks. The most important thing to recognize about a firewall is that it implements an access control policy. A firewall will typically set up a boundary between the known trustable users on one side and the potentially hackers/crackers on the other. A basic firewall setup depends greatly on the fact it is placed at the entry point to the network. If firewalls are not at every entry point of a network, they cannot block all of the unwanted traffic. Having a firewall blocking the front door is great, but you still need something watching the back door and the windows! You may have your main entrance covered, but there are often other ways into your network.

2.10.1 Packet Filtering

A packet filtering firewall is a router that is configured to validate the port and address of both the source and destination host of a packet to the control policy. Packet filtering works at the network layer of the Open System Interconnection (OSI) model or the Internet Protocol (IP) layer of the Transmission Control Protocol (TCP) TCP/IP model. The port and IP address of a packet will be matched to a control policy before allowing it onto the network.

A packet filtering system will drop all packets that do not meet the rule set you have allowed into your network. This is a quick way to set up a defense, but only serves to slow down an aggressive hacker/cracker. Packet filters are one of the easiest defenses to get around for a hacker/cracker. They do not inspect the content of the packets and are not application aware. Filters can only stop packets that come into your network through the

entry point that it is on. Setting up a packet filter at your only entry point will stop only those packets that do not meet the criteria for entry. Packet filters can and will be fooled by a savvy hacker/cracker who wants in your network.

A rule set may look similar to the following:

Type	Source Addr	Dest Addr	Source Port	Dest Port	Action
TCP	*	123.4.5.6	>1023	23	Permit
TCP	129.6.48.254	123.4.5.6	>1023	119	Permit
*	*	*	*	*	Deny

An administrator must become familiar with the different types of Internet Control Message Protocol (ICMP) packets that you may want your packet filter router to allow. There are many different types of ICMP message packets that can be identified by the type field. Here are a few examples and the Reference of each.

Type	Name	Reference
0	Echo Reply	[RFC792]
3	Destination Unreachable	[RFC792]
4	Source Quench	[RFC792]
8	Echo	[RFC792]

How Packet Filters Work

A packet filter firewall is a that analyzes network traffic at the transport protocol layer. Each IP network packet is examined to see if it matches one of a set of rules defining what data flows are allowed. These rules identify whether communication is allowed based upon information contained within the internet and transport layer headers and the direction in which the packet is headed (internal to external network or vice-versa).

Packet filters typically enable you to manipulate (that is, permit or prohibit) the transfer of data based on the following controls:

- the physical network interface that the packet arrives on
- the address the data is (supposedly) coming from (source IP address)
- the address the data is going to (destination IP address)
- the type of transport layer (TCP, UDP, ICMP)
- the transport layer source port
- the transport layer destination port

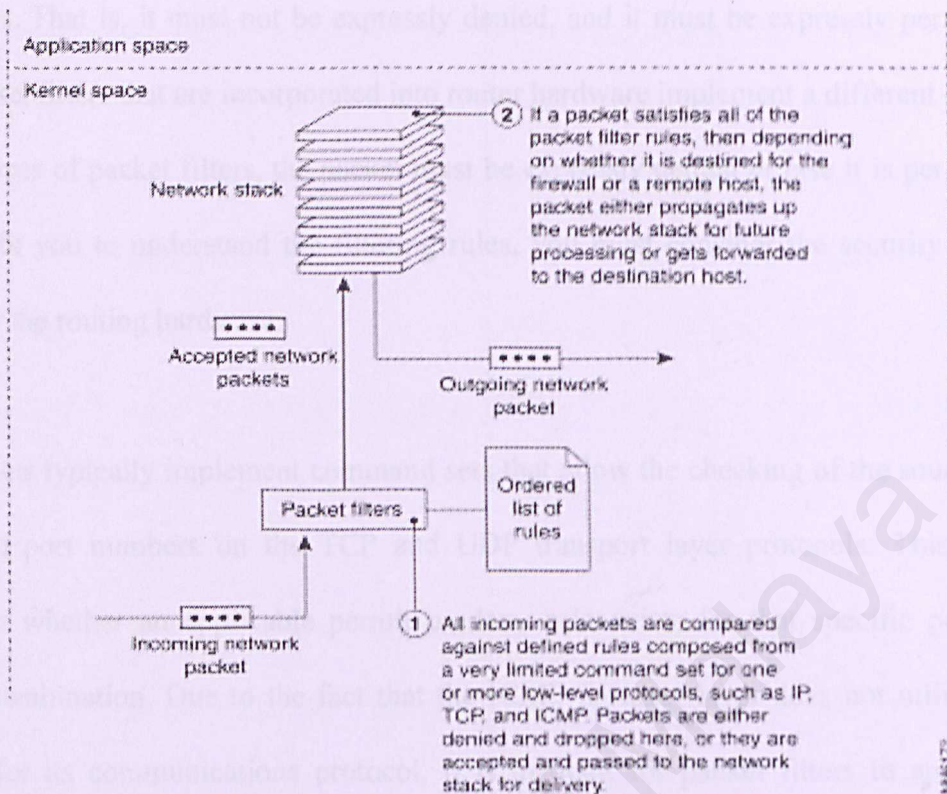


Figure 2.7: depicts the network packet evaluation process used by a packet filter firewall.

Packet filters generally do not understand the application layer protocols used in the communication packets. Instead, they work by applying a rule set that is maintained in the TCP/IP kernel. This rule set contains an associated action that will be applied to any packets matching the criteria mentioned above.

The action taken may take on one of two values: "deny" or "permit" the network packet. Two lists, the deny list and the permit list, are maintained in the kernel. For a network packet to be routed to its proper destination, it must first pass a check of both the deny and

permit lists. That is, it must not be expressly denied, and it must be expressly permitted. Some packet filters that are incorporated into router hardware implement a different policy. In these types of packet filters, the packet must be expressly denied or else it is permitted. In order for you to understand the filtering rules, you must consider the security stance utilized by the routing hardware.

Packet filters typically implement command sets that allow the checking of the source and destination port numbers on the TCP and UDP transport layer protocols. This check determines whether an applicable permit or deny rule exists for that specific port and protocol combination. Due to the fact that the ICMP protocol layer does not utilize port numbers for its communications protocol, it is difficult for packet filters to apply any security policy to this form of network traffic. In order to apply an effective security policy to ICMP, the packet filter must maintain state tables to ensure that an ICMP reply message was recently requested from an internal host. This ability to track communications state is one of the primary differences between simple packet filters and dynamic packet filters.

Because packet filters are implemented in the network layer, they generally do not understand how to process state information in the high-level protocols, such as FTP. The more sophisticated packet filters are able to detect IP, TCP, UDP, and ICMP. Using a packet filter that includes the TCP/UDP port filtering capability, you can permit certain types of connections to be made to specific computers while prohibiting other types of connections to those computers and similar connections to other computers.

Because this type of firewall does not inspect the network packet's application layer data and does not track the state of connections, this solution is the least secure of the firewall technologies. It allows access through the firewall with a minimal amount of scrutiny. In other words, if the checks succeed, the network packet is allowed to be routed through the firewall as defined by the rules in the firewall's routing table. However, because it does less processing than the other technologies, it is the fastest firewall technology available and is often implemented in hardware solutions, such as IP routers.

Packet filter firewalls often readdress network packets so that outgoing traffic appears to have originated from a different host rather than an internal host. The process of readdressing network packets is called network address translation. Network address translation hides the topology and addressing schemes of trusted networks from untrusted networks.

To summarize, firewalls based on the packet filtering technologies have the following advantages:

- Packet filters are generally faster than other firewall technologies because they perform fewer evaluations. Also, they can easily be implemented as hardware solutions.
- A single rule can help protect an entire network by prohibiting connections between specific Internet sources and internal computers.
- Packet filters do not require client computers to be specifically configured; the packet filters do all of the work.

- In conjunction with network address translation, you can use packet filter firewalls to shield internal IP addresses from external users.

Firewalls based on the packet filtering technologies have the following disadvantages:

- Packet filters do not understand application layer protocols. They cannot restrict access to protocol subsets for even the most basic services, such as the PUT or GET commands in FTP. For this reason, they are less secure than application layer and circuit level firewalls.
- Packet filters are stateless in that they do not keep information about a session or application-derived information.
- Packet filters have very limited abilities to manipulate information within a packet.
- Packet filters do not offer value-added features, such as HTTP object caching, URL filtering, and authentication because they do not understand the protocols being used and cannot discern one from another.
- Packet filters cannot restrict what information is passed from internal computers to services on the firewall server. Packet filters only restrict what information can go to it. Thus, intruders can potentially access the services on the firewall server.
- Packet filters have little or no audit event generation and alerting mechanisms.
- Because of the complexity of supporting most non-trivial network services, it can be difficult to test "accept" and "deny" rules.

2.10.2 Circuit Level Gateways

A circuit level firewall is a technology that validates the fact that a packet is either a connection request or a data packet belonging to a connection, or virtual circuit, between two peer transport layers.

To validate a session, a circuit level firewall examines each connection setup to ensure that it follows a legitimate handshake for the transport layer protocol being used (the only widely used transport protocol that uses a handshake are TCP). In addition, data packets are not forwarded until the handshake is complete. The firewall maintains a table of valid connections (which includes complete session state and sequencing information) and lets network packets containing data pass through when network packet information matches an entry in the virtual circuit table. Once a connection is terminated, its table entry is removed, and that virtual circuit between the two peer transport layers is closed.

How Circuit Level Firewalls Work

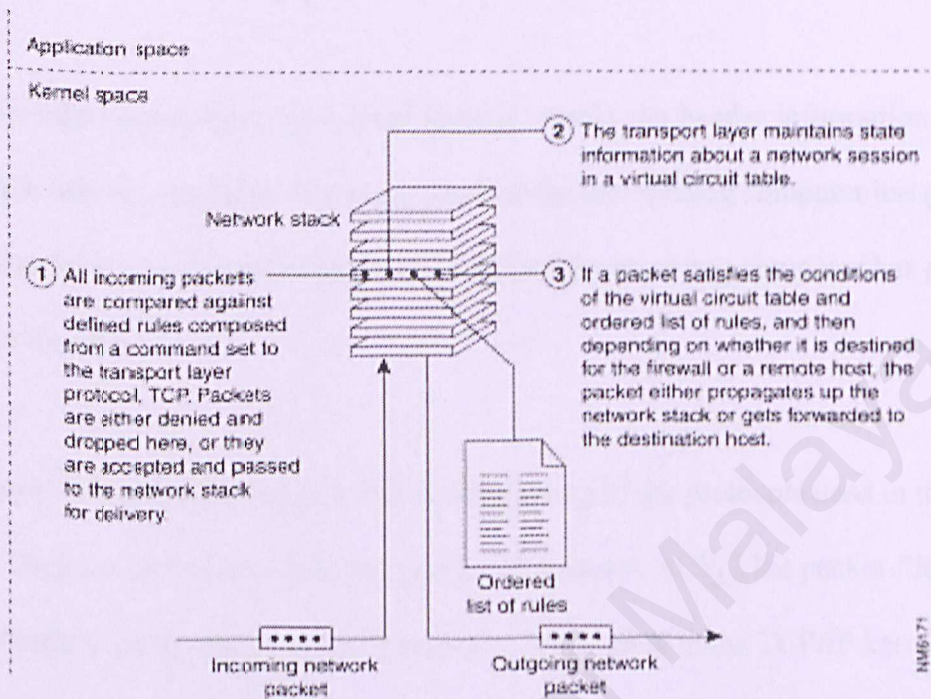


Figure 2.8: depicts the network packet evaluation process used by a circuit level firewall.

When a connection is set up, the circuit level firewall typically stores the following information about the connection:

- A unique session identifier for the connection, which is used for tracking purposes
- The state of the connection: handshake, established, or closing
- The sequencing information
- The source IP address, which is the address from which the data is being delivered
- The destination IP address, which is the address to which the data is being delivered

- The physical network interface through which the packet arrives
- The physical network interface through which the packet goes out

Using this information, the circuit level firewall checks the header information contained within each network packet to determine whether the transmitting computer has permission to send data to the receiving computer and whether the receiving computer has permission to receive that data.

Circuit level firewalls have only limited understanding of the protocols used in the network packets. They can only detect one transport layer protocol, TCP. Like packet filters, circuit level firewalls work by applying a rule set that is maintained in the TCP/IP kernel.

Circuit level firewalls allow access through the firewall with a minimal amount of scrutiny by building a limited form of connection state. Only those network packets that are associated with an existing connection are allowed through the firewall. When a connection establishment packet is received, the circuit level firewall checks its rule bases to determine whether that connection should be allowed. If the connection is allowed, all network packets associated with that connection are routed through the firewall as defined in the firewall server's routing table with no further security checks. This method is very fast and provides a limited amount of state checking.

Circuit level firewalls can perform additional checks to ensure that a network packet has not been spoofed and that the data contained within the transport protocol header complies

with the definition for that protocol, which allows the firewall to detect limited forms of modified packet data.

Circuit level firewalls often readdress network packets so that outgoing traffic appears to have originated from the firewall rather than an internal host. As stated previously, this process of readdressing network packets is called network address translation, and because circuit level firewalls maintain information about each session, they can properly map external responses back to the appropriate internal host.

To summarize, circuit level firewalls have the following advantages:

- Circuit level firewalls are generally faster than application layer firewalls because they perform fewer evaluations.
- A circuit level firewall can help protect an entire network by prohibiting connections between specific Internet sources and internal computers.
- In conjunction with network address translation, you can use circuit level firewalls to shield internal IP addresses from external users.

Circuit level firewalls have the following disadvantages:

- Circuit level firewalls cannot restrict access to protocol subsets other than TCP.
- Circuit level firewalls cannot perform strict security checks on a higher-level protocol should the need arise.
- Circuit level firewalls have limited audit event generation abilities but can typically tie a network data packet to an application layer protocol by building limited forms of session state.

- Circuit level firewalls do not offer value-added features, such as HTTP object caching, URL filtering, and authentication because they do not understand the protocols being used and cannot discern one from another.
- It can be difficult to test "accept" and "deny" rules.

University of Malaya

2.10.3 Application Layer Firewalls

An application layer firewall is a technology that evaluates network packets for valid data at the application layer before allowing a connection. It examines the data in all network packets at the application layer and maintains complete connection state and sequencing information. In addition, an application layer firewall can validate other security items that only appear within the application layer data, such as user passwords and service requests.

Most application layer firewalls include specialized application software and proxy services. Proxy services are special-purpose programs that manage traffic through a firewall for a specific service, such as HTTP or FTP. Proxy services are specific to the protocol that they are designed to forward, and they can provide increased access control, careful detailed checks for valid data, and generate audit records about the traffic that they transfer.

How Application Layer Firewalls Work

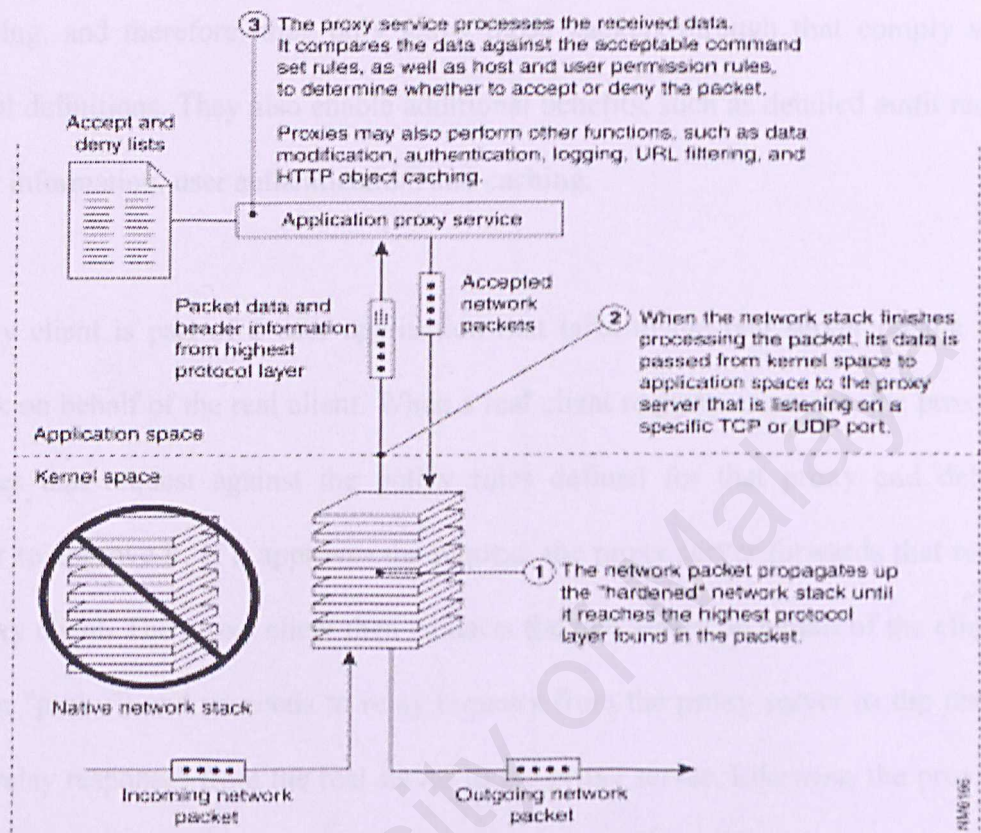


Figure 2.9: depicts the network packet evaluation process used by a application layer firewall.

Each application proxy requires two components that are typically implemented as a single executable: a proxy server and a proxy client. A proxy server acts as the end server for all connection requests originated on a trusted network by a real client. That is, all communication between internal users and the Internet passes through the proxy server rather than allowing users to communicate directly with other servers on the Internet. An internal user, or client, sends a request to the proxy server for connecting to an external service, such as FTP or Telnet. The proxy server evaluates the request and decides to

permit or deny the request based on a set of rules that are managed for the individual network service. Proxy servers understand the protocol of the service that they are evaluating, and therefore, they only allow those packets through that comply with the protocol definitions. They also enable additional benefits, such as detailed audit records of session information, user authentication, and caching.

A proxy client is part of a user application that talks to the real server on the external network on behalf of the real client. When a real client requests a service, the proxy server evaluates that request against the policy rules defined for that proxy and determines whether to approve it. If it approves the request, the proxy server forwards that request to the proxy client. The proxy client then contacts the real server on behalf of the client (thus the term "proxy") and proceeds to relay requests from the proxy server to the real server and to relay responses from the real server to the proxy server. Likewise, the proxy server relays requests and responses between the proxy client and the real client.

How a Proxy Service Works

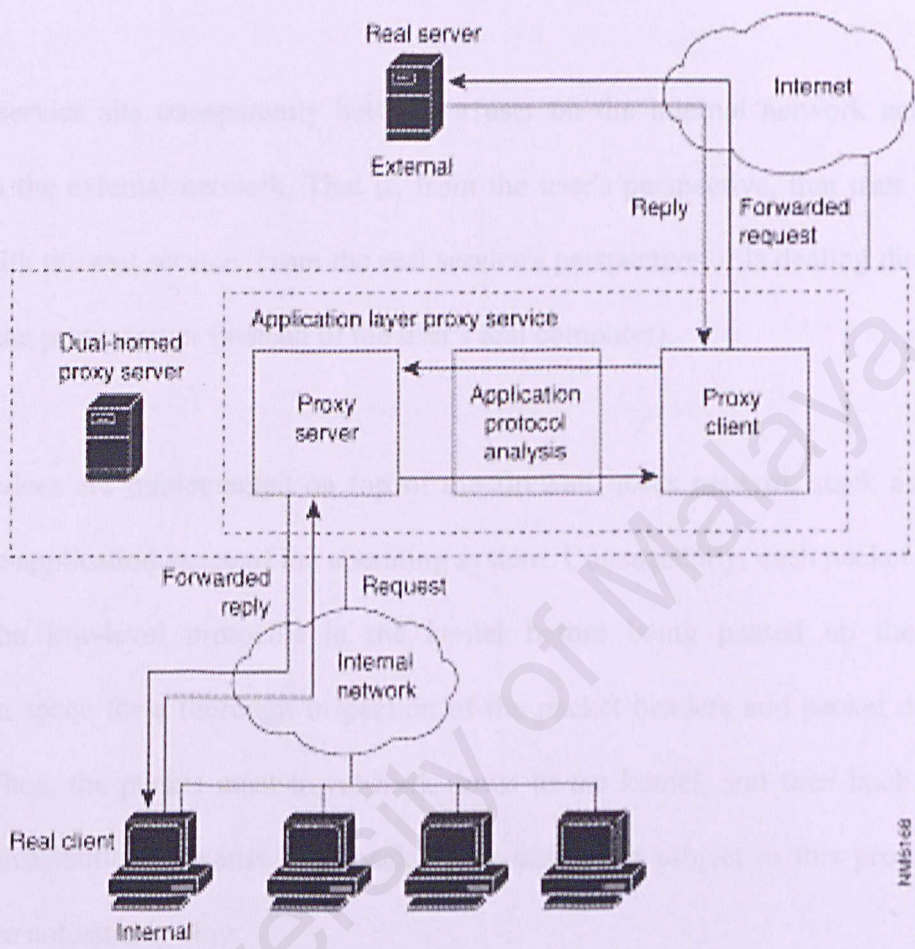


Figure 2.10: depicts the flow of communications between a real client and a network server when the communications pass through a proxy service.

Proxy services never allow direct connections, and they force all network packets to be examined and filtered for suitability. Instead of communicating directly with the real service, a user communicates to the proxy server (because the user's default gateway is set to point to the proxy server on the firewall). The same is true from the perspective of the

real service communicating with a user. The proxies handle all communications between the user and a real service.

A proxy service sits transparently between a user on the internal network and the real service on the external network. That is, from the user's perspective, that user is dealing directly with the real service. From the real service's perspective, it is dealing directly with a user on the proxy server (instead of the user's real computer).

Proxy services are implemented on top of the firewall host's network stack and operate only in the application space of the operating system. Consequently, each packet must pass through the low-level protocols in the kernel before being passed up the stack to application space for a thorough inspection of the packet headers and packet data by the proxies. Then, the packet must travel back down to the kernel, and then back down the stack for distribution. Because each packet in a session is subject to this process, proxy services are notoriously slow.

Like circuit level firewalls, application layer firewalls can perform additional checks to ensure that a network packet has not been spoofed, and they often perform network address translation.

To summarize, proxy services have several key advantages:

- Proxy services understand and enforce high-level protocols, such as HTTP and FTP.

- Proxy services maintain information about the communications passing through the firewall server. They provide partial communication-derived state information, full application-derived state information, and partial session information.
- Proxy services can be used to deny access to certain network services, while permitting access to others.
- Proxy services are also capable of processing and manipulating packet data.
- Proxy services do not allow direct communications between external servers and internal computers, so the names of internal computers do not have to be made known to external computers. In other words, proxy services shield internal IP addresses from the external world.
- By providing transparency, proxies provide users with the appearance that they are communicating directly with external servers.
- Proxy services can route internal services, as well as external-to-internal requests, elsewhere (for example, they can route services to an HTTP server on another computer).
- Proxy services can provide value-added features, such as HTTP object caching, URL filtering, and user authentication.
- Proxy services are good at generating audit records, allowing administrators to monitor attempts to violate the firewall's security policies.

Proxy services also have some disadvantages. These disadvantages include the following:

- Proxy services require you to replace the native network stack on the firewall server.

- Because the proxy servers listen on the same port as network servers, you cannot run network servers on the firewall server.
- Proxy services introduce performance delays. Inbound data has to be processed twice, by the application and by its proxy (for example, the Internet e-mail application talks to the proxy e-mail agent, which in-turn talks to a LAN e-mail application).
- Generally, a new proxy must be written for each protocol that you want to pass through the firewall, and therefore, the number of available network services and their scalability is limited. Usually a lag of six months or more exists from when the application is available and when its proxy is available, meaning users must wait for mission-critical applications to be available to them.
- Application level firewalls cannot provide proxies for UDP, RPC, and other services from common protocol families.
- Proxy services often require modifications to clients or client procedures, thus adding a task to the configuration process.
- Proxy services are vulnerable to operating-system and application-level bugs. Most packet filter firewalls do not rely extensively on operating system support mechanisms; however, they do generally rely on device drivers, etc. Most application layer firewalls require extensive support from the operating system to operate correctly, such as support from NDIS, TCP/IP, WinSock, Win32, and the standard C library. If a security relevant bug appears in any of these libraries, it can have undesirable effects on the security of the firewall server.
- Application layer firewalls overlook network packet information that is contained in lower layers. If the network stack is not performing correctly (which is complex

to validate), then some of the information used to perform security checks that application layer firewalls request using standard calls from operating system libraries could return incorrect information. An example call that is often utilized by application layer firewalls is the `getpeeraddress()` call.

- Proxies may require additional passwords or other validation procedures that introduce delays and frustrate users.

2.10.4 Dynamic Packet Filters /Stateful Inspection

A dynamic packet filter firewall is a technology that allows modification of the security rule base on the fly. This type of technology is most useful for providing limited support for the UDP transport protocol. The UDP transport protocol is typically used for limited information requests and queries in application layer protocol exchanges.

This firewall accomplishes its functional requirements by associating all UDP packets that cross the security perimeter with a virtual connection. If a response packet is generated and sent back to the original requester, then a virtual connection is established and the packet is allowed to traverse the firewall server. The information associated with a virtual connection is typically remembered for a short period of time, and if no response packet is received within this time period, the virtual connection is invalidated.

Network packet evaluation process

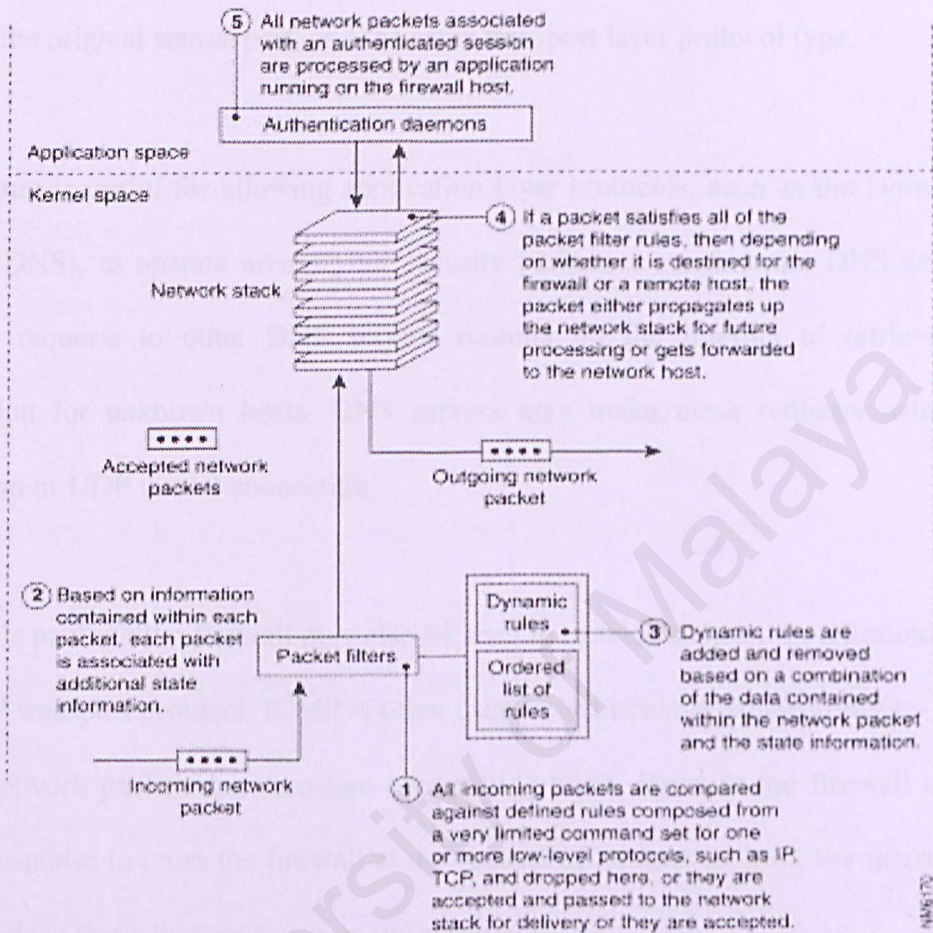


Figure 2.11: depicts the network packet evaluation process used by a dynamic packet filter firewall.

Dynamic packet filter firewalls have the same advantages and disadvantages associated with first-generation packet filter firewalls with one notable exception: the advantage of not allowing unsolicited UDP packets onto your internal network. As long as a UDP request packet originated on your internal network and is delivered to an untrusted host, the firewall server allows what appears to be a response packet to be delivered to the

originating host. The response packet that is allowed back must contain a destination address that matches the original source address, a transport layer destination port that matches the original source port, and the same transport layer protocol type.

This feature is useful for allowing application layer protocols, such as the Domain Name System (DNS), to operate across your security perimeter. An internal DNS server must originate requests to other DNS servers running on the Internet to retrieve address information for unknown hosts. DNS servers may make these requests using a TCP connection or UDP virtual connection.

A dynamic packet filter firewall may also be used to provide support for a limited subset of the ICMP transport protocol. ICMP is often used to test network connectivity by sending a pair of network packets between two cooperating hosts. Because the firewall server can allow a response to cross the firewall at the request of an internal host, the internal host is able to deduce that a host exists on an untrusted network.

CHAPTER 3

METHODOLOGY

3.1 Overview

This chapter will define the Methodology used for FirePower development. A Methodology will offer a step-by-step approach to the desired situation – and each step will offer some success and benefit. Which doesn't deliver any real benefits before it is fully implemented is not made for real life.

In order to achieving high-quality architectural designs in software development are, the following items should take in consideration:

- awareness of the importance of architectural design to software development
- understanding of the role of the software architect
- understanding of the design process
- design experience in a development organization
- sufficient software architecture design methods and tools
- understanding of how to evaluate designs
- communication among stakeholders

3.2 Process Model

There are some process models:

- I. Waterfall Model
- II. V Model
- III. System Development Life Cycle (SDLC)
- IV. Spiral Model
- V. Iterative-and-Incremental Model

3.2.1 Iterative-and-Incremental Model

Iterative-and-Incremental Model has been used to develop FirePower because:

- There are multiple opportunities for checking that the software product is correct
- Every iteration incorporates the test workflow
- Faults can be detected and corrected early
- The robustness of the architecture can be determined early in the life cycle
- Architecture — the various component modules and how they fit together
- Robustness — the property of being able to handle extensions and changes without falling apart
- Scope of project well understand
- Project risks have been accessed and are considered to be low.

Prototyping is a sub-process and prototype is a partially developed product or a simple simulator of the actual system to examine the proposed system and overview on

the functionalities. A prototype of FirePower will be built regarding to the project scope and the analysis of the system before start to build the actual system.

Prototyping is very important because:

- To ensure the system meet the performance goals or constraints.
- To ensure the system are practical and flexible.
- To ensure the system fulfill the users' requirement.
- To have an insight of how the module and sub-modules interact with each other.

Iterative-and-Incremental Model

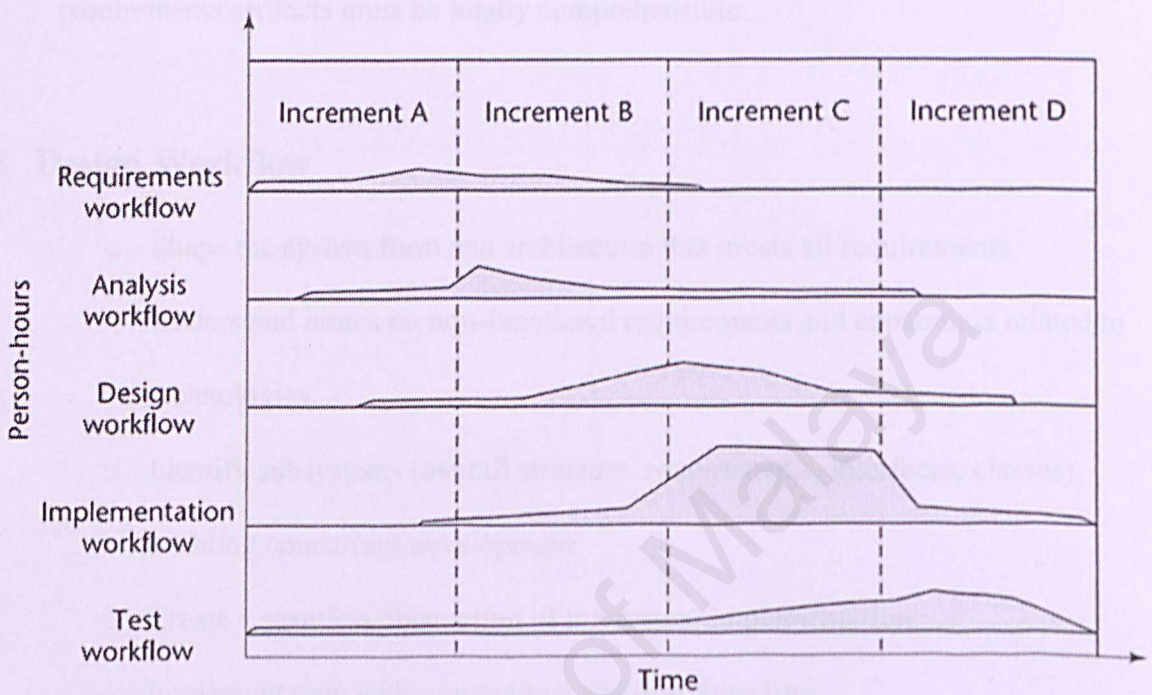


Figure 3.1: Iterative-and-Incremental Model

3.3 Requirements Workflow

To determine the computer security's needs

- i. First, gain an understanding of the application domain. That is, the specific environment in which the software product is to operate.
- ii. Second, build a model Use UML to describe the processes .If at any time the cost is not justified, development terminates immediately.

3.4 Analysis Workflow

- i. The aim of the analysis workflow is to analyze and refine the requirements. The requirements artifacts must be totally comprehensible.

3.5 Design Workflow

- Shape the system form and architecture that meets all requirements.
- Understand issues on non-functional requirements and constraints related to technologies.
- Identify subsystems (overall structure, requirements, interfaces, classes) enabling concurrent development.
- Create a seamless abstraction of the system implementation.
- Implementation adds content to a stable architecture.
- Provides a visualization of the implementation.
- Architectural design :
- Consider patterns and implementation reuse.
- Identify nodes and network configurations.
- Nodes: number and processing power.
- Connections: protocols, bandwidth, quality, etc.
- Fault tolerance: redundancy, fail-over, migration, data backup, etc.
- Identify subsystems and their interfaces.
- First iteration is analysis package --> subsystem.
- Refine for shared functionality, reused/wrapped legacy software, load balancing, etc.
- Define dependencies and layers.

- Define interfaces to serve dependencies.
- When designing use cases in terms of subsystems and interfaces, identify operations on each interface.
- Identify generic design mechanisms: persistence, transparent object distribution, security, error detection and recovery, transaction management, etc.

2.6 Implementation Workflow

- The two phases of the program implementation process:
 - i. The delivery of the program.
 - ii. How the program and the infrastructure are modified to adjust to the evolution of the market.

2.7 Testing Workflow

- Testing of individual program components
- Usually the responsibility of the component developer (except sometimes for critical systems)
- Testing of groups of components integrated to create a system or sub-system
- Tests are based on a system specification

CHAPTER 4 SYSTEM REQUIREMENTS ANALYSIS

The need for Internet firewalls has increased tremendously over the years. Everyone is looking for the best firewall solutions to protect the internal components of their networks from threats and network intrusions. If the needs of the organization are not well defined, then it is almost impossible to identify the best firewall solution. Therefore, first must determine the needs of machine and then design a firewall solution that best meets those needs.

This chapter explains what to build firewalls using different technologies that are available. It also presents the evaluation criteria should apply when developing the most appropriate firewall.

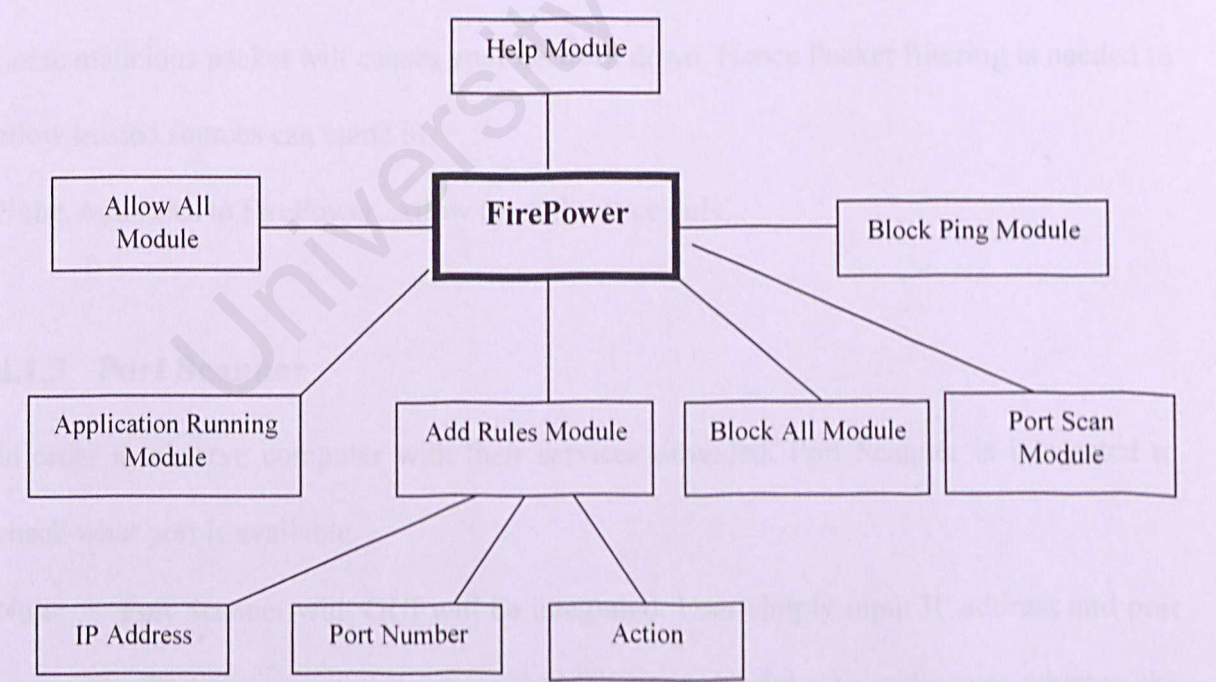


Figure 4.1: Components of FirePower

4.1 Functional Requirements

The following will illustrates all functions and policies will be provided at FirePower. Now we come to see how does the security threat been solved by FirePower.

4.1.1 Block Ping

The Ping of Death is the granddaddy of all denial-of-service attacks. It exploits the fact that many TCP/IP implementations trust that ICMP packets are correctly formed and perform too little error checking.

Note: Configuring firewalls to block ICMP and any unknown protocols will prevent this attack.

4.1.2 Packet Filtering

Some malicious packet will causes your network down. Hence Packet filtering is needed to allow trusted sources can come in.

Note: Add rules to FirePower. Allow trusted source only.

4.1.3 Port Scanner

In order to observe computer with their services provided. Port Scanner is integrated to check what port is available.

Note: A Port Scanner with GUI will be integrated. User simply input IP address and port range. The Port Scanner will do an analysis to your computer to make sure whether the specify port is open or not.

4.1.4 Generate Log file

A successful Personal Firewall not only able to defend the malicious packet but also can do some sample accounting management.

Note: A Log file will be generating automatically to let the user know what packet has been blocked.

4.1.5 GUI

In order to simplify the usage of FirePower, a user friendly interface is required. Every function is executed just by a click.

Note: The user friendly interface let user to manage the personal firewall setting. Thus, it will make the life easier.

4.1.6 Help Menu

User may face some difficulties when using the FirePower. A clear guideline may teach them how to use.

Note: A briefly guideline is provided to help new user to learn how to use the FirePower.

4.1.7 System Tray

It is a necessary a shortcut being created while running the system. It makes the window look more neatly.

Note: Its better way to hide the GUI and then call it back when there is the needs.

4.2 Non-Functional Requirement

Reliability: The system should be reliable in performing its protection function and network operation. For example, whenever a button is clicked, the system should be able to perform some functionality or generate some message or animation to inform the user what is happening.

Usability: The system should be user friendly. User must be able to learn how to use the system in the shortest period. It will enhance and support rather than limit or restrict the understanding of firewall. Human interfaces need to be intuitive and consistent with the user knowledge in order to them gain some knowledge through the FirePower.

Manageability: The modules within the system should be easy to manage. This will make the maintaining, enhancement work simpler, and not time consumed. The system should not cause any damages to the current FirePower after a new component has been added. The system should be designed systematically so that the effort required to locate and fix an error in the system is minimum.

Flexibility: The system has its capabilities to make advantages of new technologies, resource and in fast changing environment. The system should be able to implement in a changing environment of platform.

Correctness: The final application must meet the objective, specification and requirement of the users.

4.3 Software and Hardware Selection

The following is a software requirement for the development of the FirePower.

Visual Studio .NET

Visual Studio .NET is a complete set of development tools for building ASP Web applications, XML Web services, desktop applications, and mobile applications. Visual Basic .NET, Visual C++ .NET, Visual C# .NET, and Visual J# .NET all use the same integrated development environment (IDE), which allows them to share tools and facilitates in the creation of mixed-language solutions. In addition, these languages leverage the functionality of the .NET Framework, which provides access to key technologies that simplify the development of ASP Web applications and XML Web services.

This section contains information about some of the latest tools and technologies available in this release of Visual Studio.

Windows Forms

Windows Forms is the new platform for Microsoft Windows application development, based on the .NET Framework. This framework provides a clear, object-oriented, extensible set of classes that enables you to develop rich Windows applications. Additionally, Windows Forms can act as the local user interface in a multi-tier distributed solution.

4.4 Development Requirements

The following are the hardware requirements for the development of the FirePower.

a) Hardware requirements:

- PC with at least Pentium III 600 MHz processor
- At least 128 MB of Random Access Memory (RAM)
- At least 10GB of hard disk space.
- Network Interface card.
- Other standard computer peripherals.

Software requirements:

- Microsoft Windows 2000 Professional Edition .
- Microsoft Visual Studio .NET
- Microsoft .NET Framework

User Requirements

The following are the hardware and software requirements for the user in order to use UM Peer.

a) Client Hardware Requirements

- PC with at least Pentium 166 MHz and above
- At least 64 MB of Random Access Memory (RAM)
- Network interface card.
- At least 10 GB hard disk space
- Other standard computer peripherals

CHAPTER 5 SYSTEM DESIGN

In this chapter, I will get a detailed overview of Internet firewall design for the purpose of producing a firewall solution for our machine.

After the stage of system analysis, the implementation of the entire system can be design according to the system's requirement. The object-oriented methodology will be used at the implementation phase of the development process through the C++ programming language.

5.1 Deployment Diagram

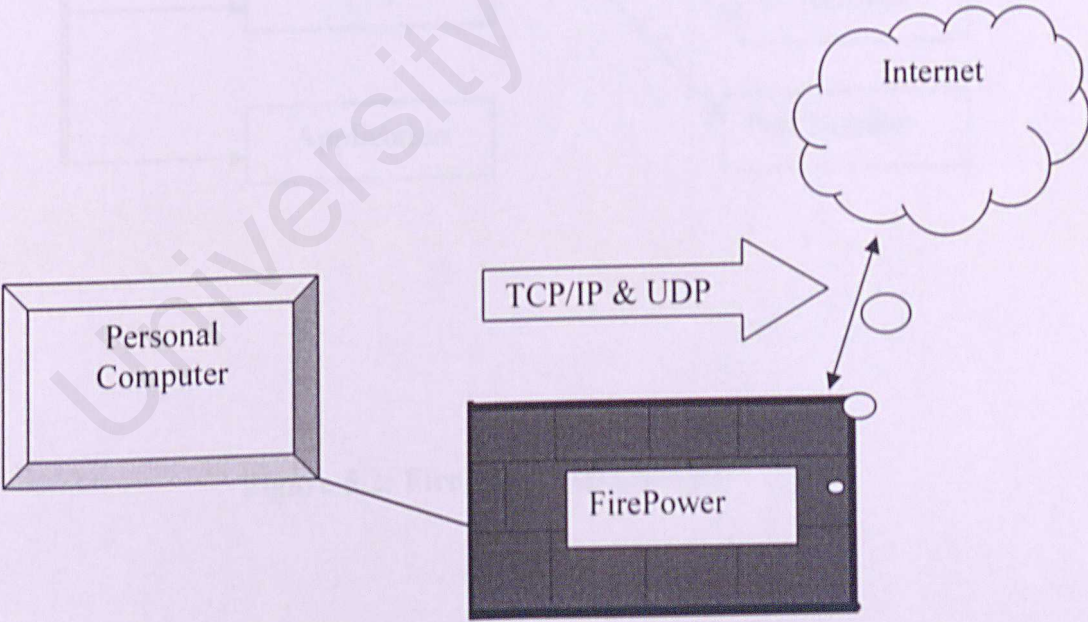


Figure 5.1: an Overview of the NetDefender is Placed

5.2 FirePower Design Overview

The overview of FirePower design is important before proceed to development of entire system.

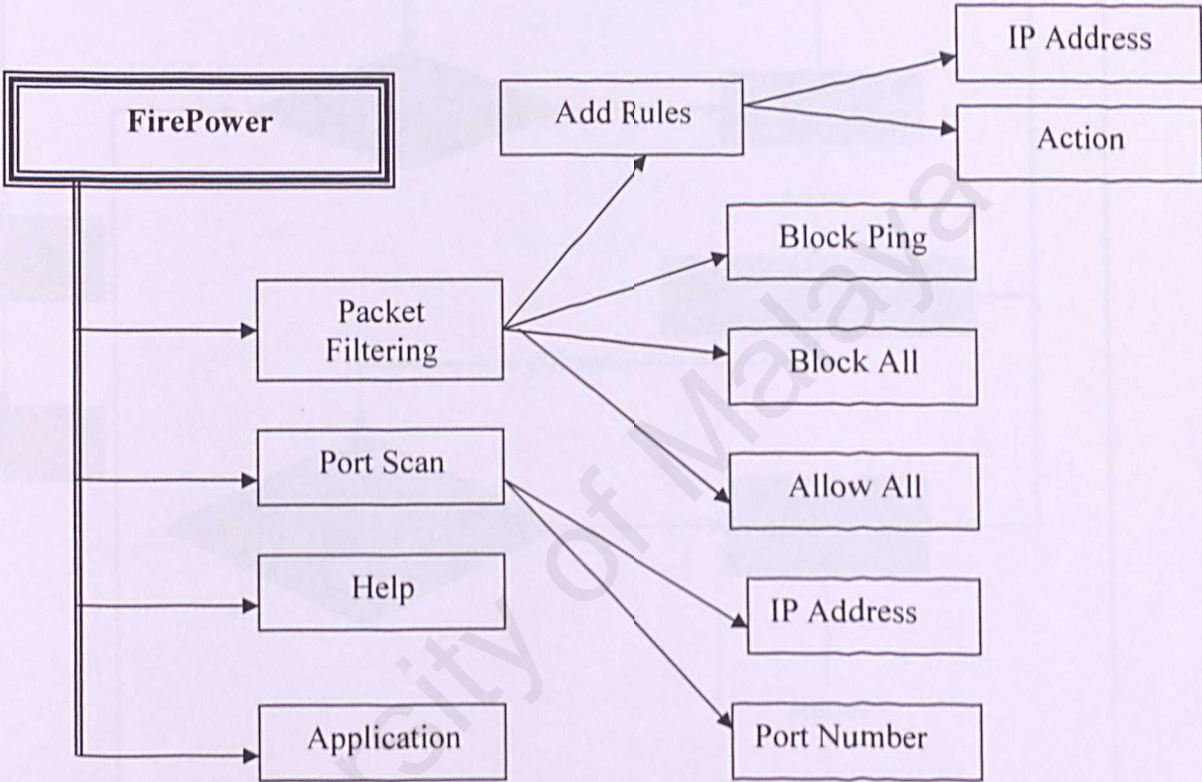


Figure 5.2: FirePower Architecture

5.3 FirePower Workflow

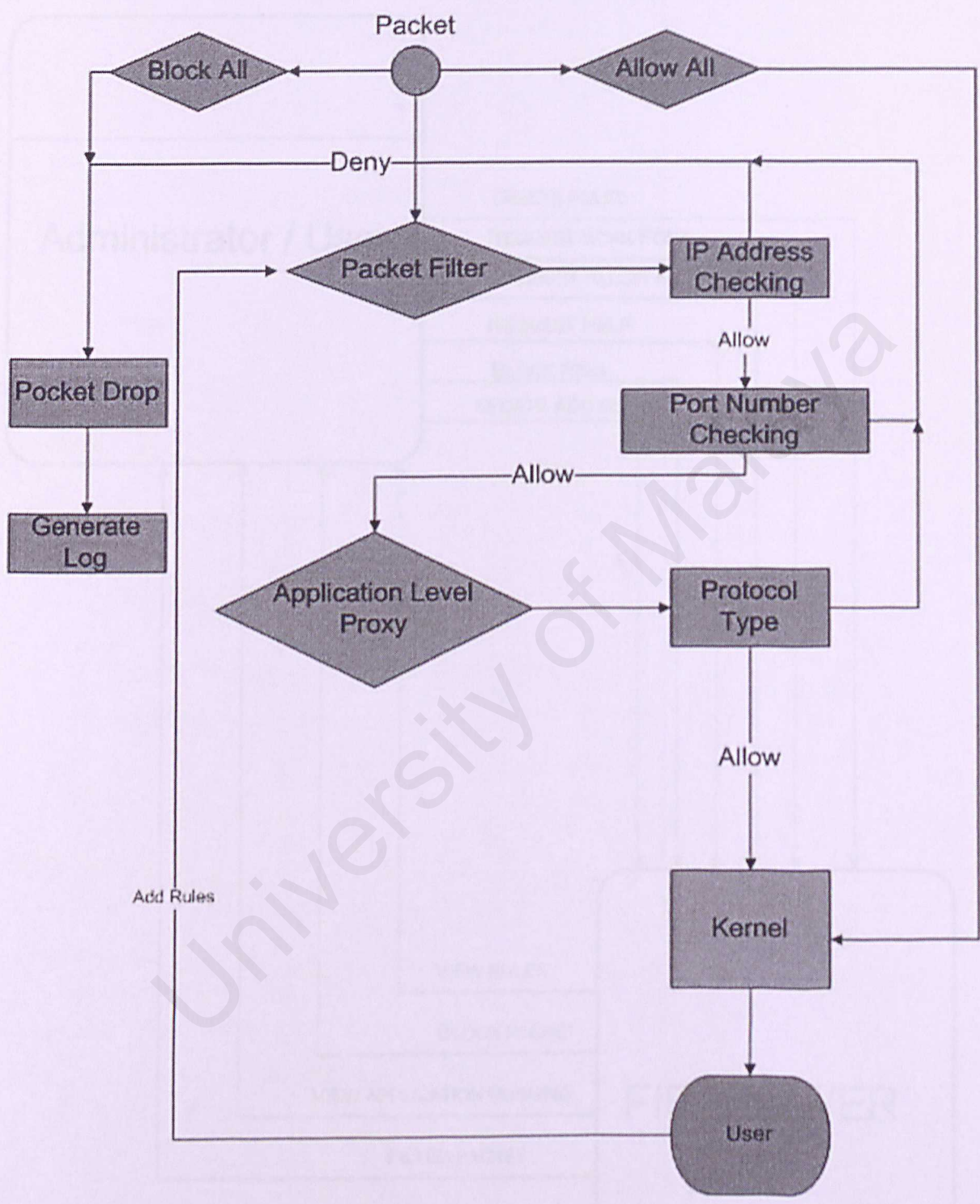


Figure 5.3: Showing FirePower Workflow

5.4 Context Diagram

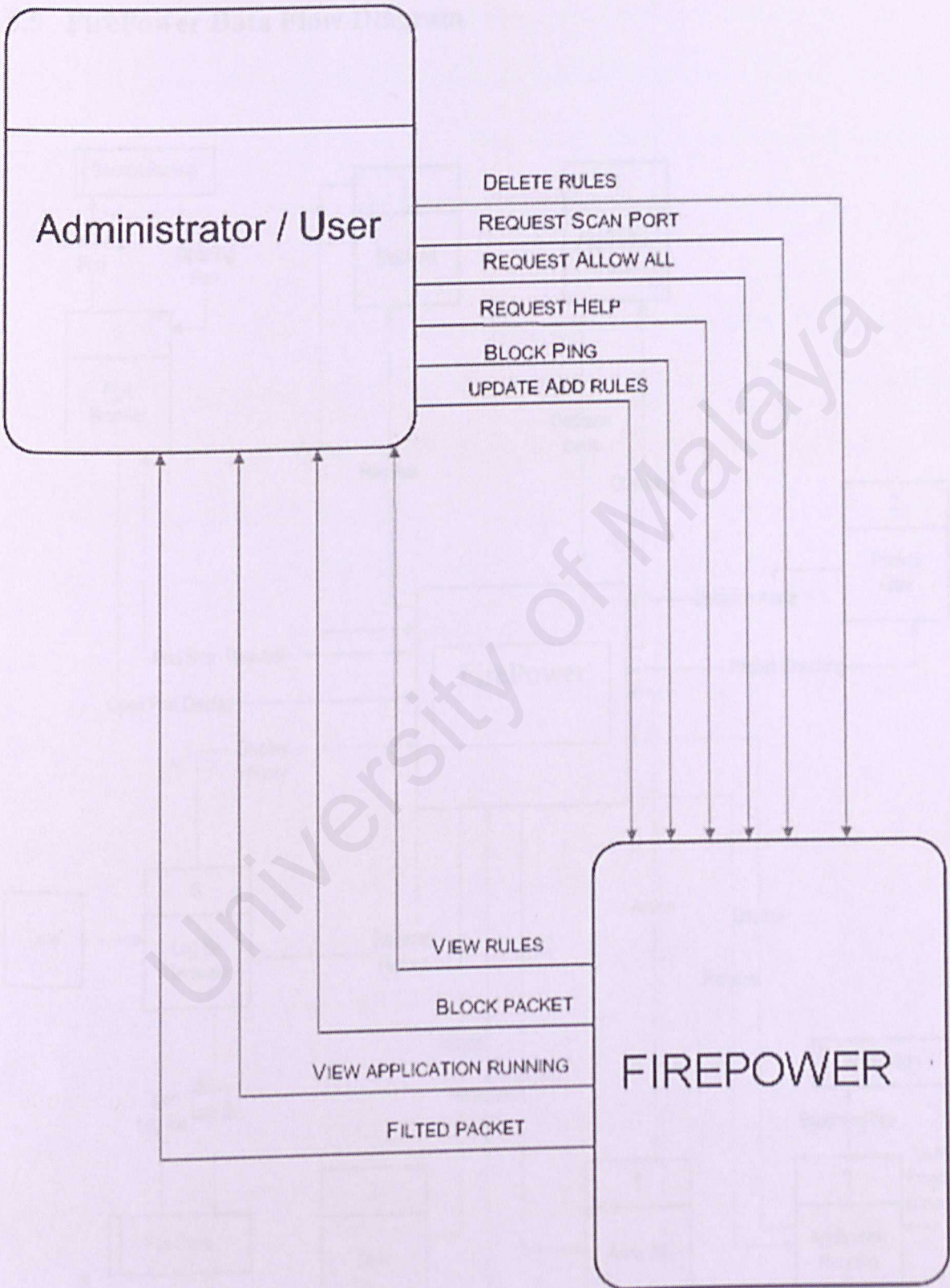


Figure 5.4: FirePower Context Diagram

5.5 FirePower User Interface Design

5.5 FirePower Data Flow Diagram

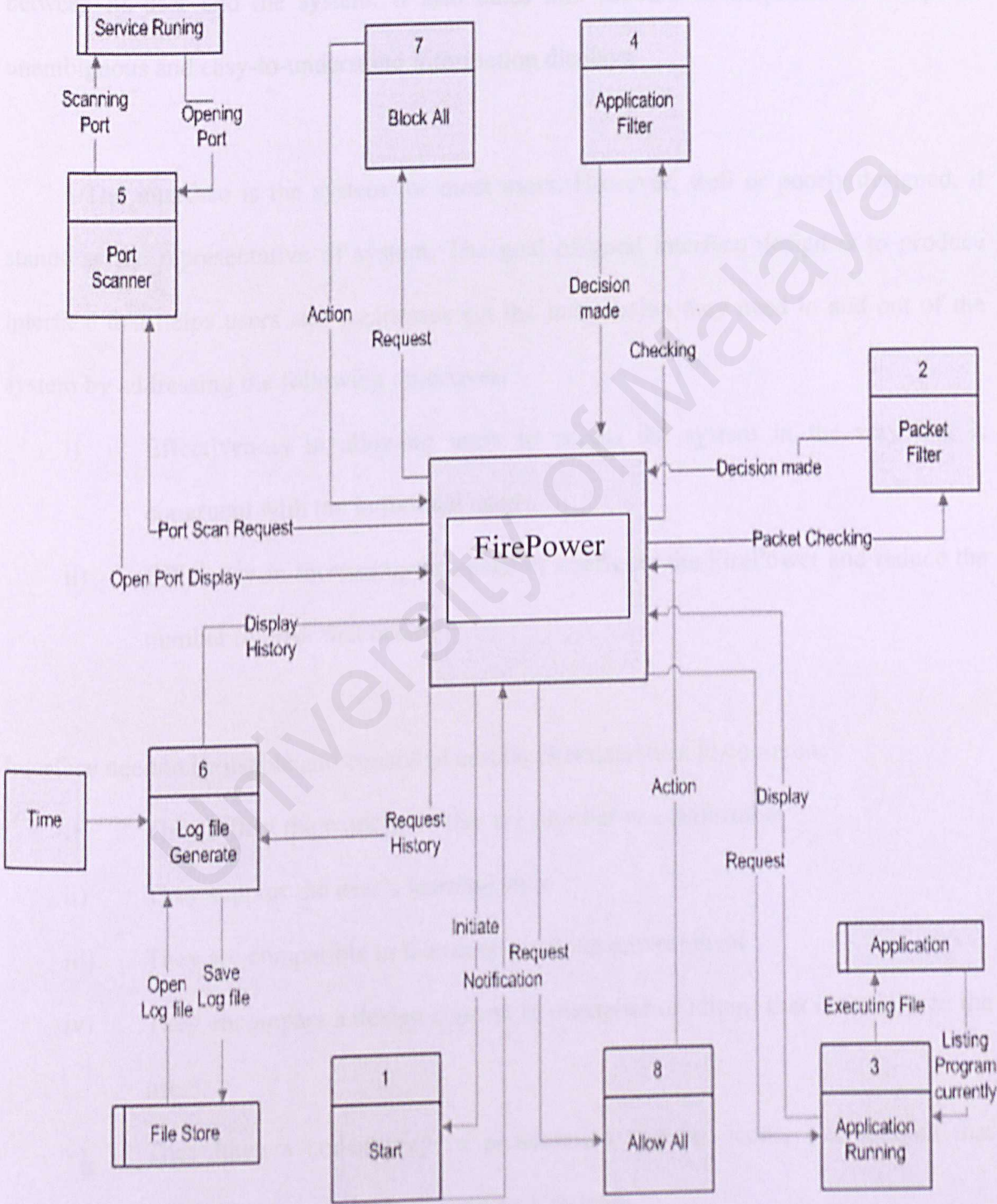


Figure 5.5: FirePower Data Flow Diagram

5.6 FirePower User Interface Design

User interface design describes how software communicates with the human user who uses it (Mundher, 1994). The user interface design focuses on the effective general interaction between its user and the system. It also takes into account development of complete, unambiguous and easy-to-understand information displays.

The interface is the system for most users. However, well or poorly designed, it stands as the representative of system. The goal of good interface design is to produce interface that helps users and businesses get the information they need in and out of the system by addressing the following objectives.

- i) Effectiveness in allowing users to access the system in the way that is congruent with the individual needs.
- ii) Efficiency in increasing the speed of configure the FirePower and reduce the number of error that occur.

Interface need to be usable and consist of certain characteristics in common:

- i) They reflect the workflows that are familiar or comfortable
- ii) They support the user's learning style
- iii) They are compatible in the users' working environment
- iv) They encompass a design concept (a metaphor or idiom) that is familiar to the users.
- v) They have a consistency of presentation (layout, icons, interactions) that makes them appear reliable and easy to learn.

- vi) The usage of languages and illustrations are familiar to the users or are easy to learn

In short, usable interfaces fit in, simply and elegantly, with users's life and work needs.

Example:

FirePower Interface Prototype Print Screen

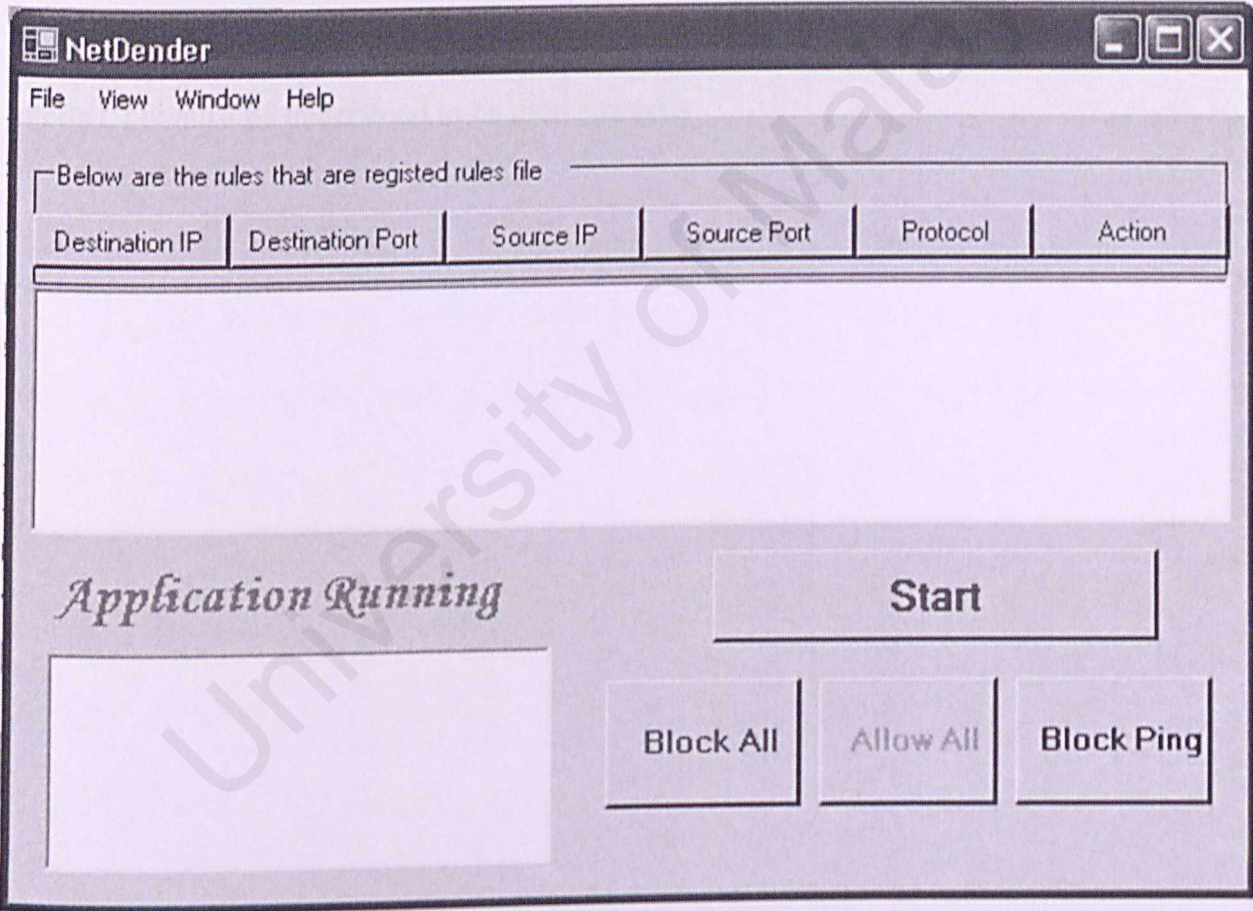


Figure 5.6: Initial NetDenfender Interface

SUMMARY

The Internet has made large amounts of information available to the average computer user at home, in business and in education. For many people, having access to this information is no longer just an advantage, it is essential. Yet connecting a private network to the Internet can expose critical or confidential data to malicious attack from anywhere in the world. Users who connect their computers to the Internet must be aware of these dangers, their implications and how to protect their data and their critical systems. Firewalls can protect both individual computers and corporate networks from hostile intrusion from the Internet, but must be understood to be used correctly.

5.1 Development Environment

In order to optimize the development process, suitable development environment is major factor to be considered. The configurations of Professor are detailed in the following table:

Development Tools	Version	Description
Microsoft Windows	Developer Platform	Operating System
Microsoft Visual Studio Professional	Programming Tool	System Development Tools
Adobe Photoshop 7.0	Image Creation Tool	Image Images for Publishing
Adobe Acrobat Workshop 4.0	PDF Creation Tool	PDF files in PDF format
Flash Help Information	Help File Creation Tool	Help Files for Help

CHAPTER 6

SYSTEM IMPLEMENTATION

6.1 Introduction

After the system –designing phase on the how the system should be functioning, the next process will include the system implementation phase. The system implementation phase is an important element especially when integration of system is needed between subsystems. In this phase, we have to consider about the issues of settings up the development environment, which include software and hardware requirements.

6.2 Development Environment

In order to optimize the development progress, suitable development environment is major factor to be considered. The software configurations of FirePower are described in the following table:

Development Tools	Function	Description
Microsoft Windows XP	Development Platform	Operating System
Microsoft Visual Studio .Net Professional	Programming Tool	System Development Tools
Adobe Photoshop 7.0	Image Creation Tool	Create images for FirePower
Axialis Icon Workshop 5.02	Icon Creation Tools	Create icons for FirePower
Fast-Help International	Help Files Creation Tools	Create Help file for FirePower

Table 6-1 List of FirePower Development Tools

6.3 Algorithms

The program design often specifies a class of algorithms to be used in the coding the component developers are writing. There are several ways developer used to organize the codes.

- Keeping the program simple and tidy to make sure that the codes are readable.
- Using data structures to determine program structures
 - Coding such as functions that needed to call for few times may put out of the main program. Besides, variables that may need to use in several forms, should put in a general module (public).
- Always give comment on code, additional comments are useful wherever helpful information can be added to a component.
 - Comment block in C++ .NET may looks like `“//This is the sample of comment block”` or `“/* This is the sample of comment block*/`.

6.4 Coding

6.4.1 The Filter-Hook Driver

We can summarize them in the following steps:

1. Create a Filter-Hook Driver. For this, we must create a Kernel Mode Driver, we choose the name, DOS name and other driver characteristics, nothing obligatory but it is recommended using descriptive names.
2. If we want to install the filter function, first we must get a pointer to IP Filter Driver. So, it will be the second step.

3. We already have the pointer, now we can install the filter function. We can do it by sending a specific IRP. The data passed in this "message" includes a pointer to the filter function.
4. Filtering packets.
5. When we decide to finish filtering, we must deregister the filter function. We can do it by "registering" as filter function the null pointer.

6.4.2 Create the Kernel Mode Driver

Filter-Hook driver is a Kernel Mode Driver, so if we want to do one, we have to make a Kernel Mode Driver.

The structure of the Filter-Hook driver is the typical Kernel Mode Driver structure:

A driver entry where we create the device set the standard routines in order to process IRPs (Dispatch, load, unload, create....) and create the symbolic link for communication with user applications.

The standard routines to manage IRPs. Before begin to code, think what IOCTL "export" to applications from device driver. In the sample, we implement four IOCTL Codes: START_IP_HOOK (registers the filter function), STOP_IP_HOOK (deregisters the filter function), ADD_FILTER (installs a new rule) and CLEAR_FILTER (frees all rules).

For our driver, we must implement one more function: the filter function.

It is recommended to use a program that generates the structure of a Kernel Mode Driver, so only have to put code into the generate functions.

We can see the implementation of the structure of the Driver, in the following code:

```
NTSTATUS DriverEntry(IN PDRIVER_OBJECT DriverObject,
                   IN PUNICODE_STRING RegistryPath)
{
    //....

    dprintf("DrvFltIp.SYS: entering DriverEntry\n");

    //we have to create the device

    RtlInitUnicodeString(&deviceNameUnicodeString, NT_DEVICE_NAME);
    ntStatus = IoCreateDevice(DriverObject, 0, &deviceNameUnicodeString,
                             FILE_DEVICE_DRVFLTIP, 0, FALSE, &deviceObject);

    if ( NT_SUCCESS(ntStatus) )
    {
        // Create a symbolic link that Win32 apps can specify to gain access
        // to this driver/device

        RtlInitUnicodeString(&deviceLinkUnicodeString, DOS_DEVICE_NAME);
        ntStatus = IoCreateSymbolicLink(&deviceLinkUnicodeString,
                                         &deviceNameUnicodeString);

        // create dispatch points for device control, create, close.

        DriverObject->MajorFunction[IRP_MJ_CREATE]      =
        DriverObject->MajorFunction[IRP_MJ_CLOSE]      =
        DriverObject->MajorFunction[IRP_MJ_DEVICE_CONTROL] = DrvDispatch;
        DriverObject->DriverUnload                      = DrvUnload;
    }
}
```



```

if ( !NT_SUCCESS(ntStatus) )
{
    dprintf("Error in initialization. Unloading...");
    DrvUnload(DriverObject);
}

return ntStatus;
}

NTSTATUS DrvDispatch(IN PDEVICE_OBJECT DeviceObject, IN PIRP Irp)
{ // ....

    switch (irpStack->MajorFunction)
    {

    case IRP_MJ_CREATE:

        dprintf("DrvFltIp.SYS: IRP_MJ_CREATE\n");

        break;

    case IRP_MJ_CLOSE:

        dprintf("DrvFltIp.SYS: IRP_MJ_CLOSE\n");

        break;

    case IRP_MJ_DEVICE_CONTROL:

        dprintf("DrvFltIp.SYS: IRP_MJ_DEVICE_CONTROL\n");

        ioControlCode = irpStack->Parameters.DeviceIoControl.IoControlCode;
    }
}

```

```

switch (ioControlCode)
{
    // ioctl code to start filtering
    case START_IP_HOOK:
    {
        SetFilterFunction(cbFilterFunction);

        break;
    }

    // ioctl to stop filtering
    case STOP_IP_HOOK:
    {
        SetFilterFunction(NULL);

        break;
    }

    // ioctl to add a filter rule
    case ADD_FILTER:
    {
        if(inputBufferLength == sizeof(IPFilter))
        {
            IPFilter *nf;

```



```

        nf = (IPFilter *)ioBuffer;

        AddFilterToList(nf);
    }

    break;

}

// ioctl to free filter rule list

case CLEAR_FILTER:

{
    ClearFilterList();

    break;
}

default:

    Irp->IoStatus.Status = STATUS_INVALID_PARAMETER;
    dprintf("DrvFltIp.SYS: unknown IRP_MJ_DEVICE_CONTROL\n");

    break;
}

// ioctl to free filter rule list

case CLEAR_FILTER:

{
    ClearFilterList();

    break;
}

```

default:

```
Irp->IoStatus.Status = STATUS_INVALID_PARAMETER;

dprintf("DrvFtIp.SYS: unknown IRP_MJ_DEVICE_CONTROL\n");

break;

}

break;

}

ntStatus = Irp->IoStatus.Status;

IoCompleteRequest(Irp, IO_NO_INCREMENT);

// We never have pending operation so always return the status code.

return ntStatus;

}

VOID DrvUnload(IN PDRIVER_OBJECT DriverObject)

{

    UNICODE_STRING deviceLinkUnicodeString;

    dprintf("DrvFtIp.SYS: Unloading\n");

    SetFilterFunction(NULL);

    // Free any resources

    ClearFilterList();

    // Delete the symbolic link

    RtlInitUnicodeString(&deviceLinkUnicodeString, DOS_DEVICE_NAME);

    IoDeleteSymbolicLink(&deviceLinkUnicodeString);
```



```
// Delete the device object
```

```
IoDeleteDevice(DriverObject->DeviceObject);
```

```
}
```

We already have made the driver main code, so we follow with code of the Filter-Hook Driver.

6.4.2 Registering a Filter function

In the above code, we have seen a function called `SetFilterFunction(..)`. We implemented this function to register a function in the IP Filter Driver. We will describe the steps followed:

First, we must get a pointer to the IP Filter Driver. That requires that driver is installed and executing. My user application loads and starts IP Filter Driver before loading this driver, in order to assure this.

Second, we must build an IRP specifying `IOCTL_PF_SET_EXTENSION_POINTER` as IO Control Code. We must pass as parameter a `PF_SET_EXTENSION_HOOK_INFO` structure that has information about the pointer to filter function. If we want to uninstall the function, we have to follow the same steps but passing `NULL` as the pointer to filter function.

Send the build IRP to the device driver.

Here there is one of the bigger problems of this driver. Only one filter function can be installed, so if other applications installed one, we can not install wer function.

I will show in the following lines, the code of this function:

```
NTSTATUS SetFilterFunction
(
    PacketFilterExtensionPtr filterFunction
)
{
    NTSTATUS status = STATUS_SUCCESS, waitStatus=STATUS_SUCCESS;
    UNICODE_STRING filterName;
    PDEVICE_OBJECT ipDeviceObject=NULL;
    PFILE_OBJECT ipFileObject=NULL;
    PF_SET_EXTENSION_HOOK_INFO filterData;
    KEVENT event;
    IO_STATUS_BLOCK ioStatus;
    PIRP irp;
    dprintf("Getting pointer to IpFilterDriver\n");

    //first of all, we have to get a pointer to IpFilterDriver Device
    RtlInitUnicodeString(&filterName, DD_IPFLTRDRVR_DEVICE_NAME);
    status = IoGetDeviceObjectPointer(&filterName, STANDARD_RIGHTS_ALL,
        &ipFileObject, &ipDeviceObject);
}
```



```

if(NT_SUCCESS(status))
{
    //initialize the struct with functions parameters

    filterData.ExtensionPointer = filterFunction;

    //we need initialize the event used later by
    //the IpFilterDriver to signal us
    //when it finished its work
    KeInitializeEvent(&event, NotificationEvent, FALSE);

    //we build the irp needed to establish filter function
    irp =
IoBuildDeviceIoControlRequest(IOCTL_PF_SET_EXTENSION_POINTER,
    ipDeviceObject, if(irp != NULL)
    {
        // we send the IRP
        status = IoCallDriver(ipDeviceObject, irp);

        //and finally, we wait for
        //"acknowledge" of IpFilter Driver
        if (status == STATUS_PENDING)
        {
            waitStatus = KeWaitForSingleObject(&event,
            Executive, KernelMode, FALSE, NULL);

            if (waitStatus != STATUS_SUCCESS )
                dprintf("Error waiting for IpFilterDriver response.");
        }
    }
}

```

```

status = ioStatus.Status;

if(!NT_SUCCESS(status))

    dprintf("Error, IO error with ipFilterDriver\n");

}

else

{

    //if we cant allocate the space,

    //we return the corresponding code error

    status = STATUS_INSUFFICIENT_RESOURCES;

    dprintf("Error building IpFilterDriver IRP\n");

}

if(ipFileObject != NULL)

    ObDereferenceObject(ipFileObject);

ipFileObject = NULL;

ipDeviceObject = NULL;

}

else

    dprintf("Error while getting the pointer\n");

return status;

}

```


We can see that when we finish the process of establishing the filter function, we must de-reference the file object obtained when we got a pointer to the device driver. We use an event to be notified when IpFilter Driver finish the processes of the IRP.

6.4.3 The Filter function

We have seen how we can develop the driver and how to install the filter function, but we don't know anything about this function yet.

We had said that this function is called always when the host receives or sends a packet. Depending on the return value of this function, the system decides what to do with the packet.

The prototype of this function must be:

```
typedef PF_FORWARD_ACTION
(*PacketFilterExtensionPtr)(
    // Ip Packet Header
    IN unsigned char *PacketHeader,
    // Packet. Don't include Header
    IN unsigned char *Packet,
    // Packet length. Don't include length of ip header
    IN unsigned int PacketLength,
    // Index number for the interface adapter
    //over which the packet arrived
    IN unsigned int RecvInterfaceIndex,
```

```
// Index number for the interface adapter
```

```
//over which the packet will be transmitted
```

```
IN unsigned int SendInterfaceIndex,
```

```
//IP address for the interface
```

```
//adapter that received the packet
```

```
IN IPAddr RecvLinkNextHop,
```

```
//IP address for the interface adapter
```

```
//that will transmit the packet
```

```
IN IPAddr SendLinkNextHop
```

```
);
```

PF_FORWARD_ACTION is an enumerated type that can value (in Microsoft Words):

PF_FORWARD

Specifies for the IP filter driver to immediately return the forward response to the IP stack. For local packets, IP forwards them up the stack. If the destination for packets is another computer and routing is enabled, IP routes them accordingly.

PF_DROP

Specifies for the IP filter driver to immediately return the drop response to the IP stack. IP should drop the packet.

PF_PASS

Specifies for the IP filter driver to filter packets and return the resulting response to the IP stack. How the IP filter driver proceeds to filter packets is determined by how it was set with the Packet Filtering API.

The filter hook returns this pass response if it determined that it should not process the packet but should allow the IP filter driver to filter the packet.

Although DDK documentation only include these 3 values, if look into pfhook.h (include needed for Filter-Hook Driver), we can see one more. This value is PF_ICMP_ON_DROP. Suppose this value correspond with dropping the packet and informing source for error with an ICMP packet.

As we can see in the definition of the filter function, the packet and its header are passed as pointers. So, we can modify header or payload and then forward the packets. This is very useful for example to do Network Address Translation (NAT). If we change destination address, IP routes the packets.

In our implementation, the filter function compares each packet with a list of rules, introduced by the user application. This list is implemented as a linked list that is built in runtime with each START_IP_HOOK IOCTL.

User Application: It's a MFC application that manages the filter rules. This application sends the rules to the application and decides when the driver must begin to filter. Three steps for filtering the traffic:

Define the rules we need. With Add and Delete commands we can add or delete filter rules.

Install Rules. When we define the rules, click install button to send them to the driver.

Start Filtering. We only have to click start button in order to begin filtering.

Filter-Hook Driver: Driver that filter IP Traffic based in the filter rules received from user application.

The Filter-Hook Driver must be in the same directory as the user application executable.

6.5 The reason why use this method to develop a Firewall

It is not the unique method to develop firewalls for Windows, there are others as NDIS Firewall, TDI Firewall, Winsock Layered Firewall, Packet Filtering API. So we will mention some advantages and disadvantages of Filter-Hook Driver.

We have much flexibility filtering with this method. We can filter all IP traffic (and above). However we can not filter lower layer header, for example, we can not filter Ethernet frame. We need a NDIS filter to do it, more complicated to develop but more flexible.

It is an easy method. Installing a firewall and the implementation of filter function is an easy procedure with this method. However Packet Filtering API is easier yet, although it is less flexible. We can not access packet content, and we can not modify this with Packet Filtering API.

Although this driver has not bad characteristics it has a great disadvantage. As we had mentioned this before, only one filter function can be installed each time. We can develop a great firewall, it can be downloaded and installed by thousands of users but if other

applications use this filter (and installed the filter function before) our program would not do anything.

7.4.1 Unit Testing

System testing is executed prior to ensure the system performs according to its specifications and to detect any errors, requirements and expectations. Testing is done throughout the life of the project and not just at the end. The following figure 7-4 is showing the progression of system testing and delivery.

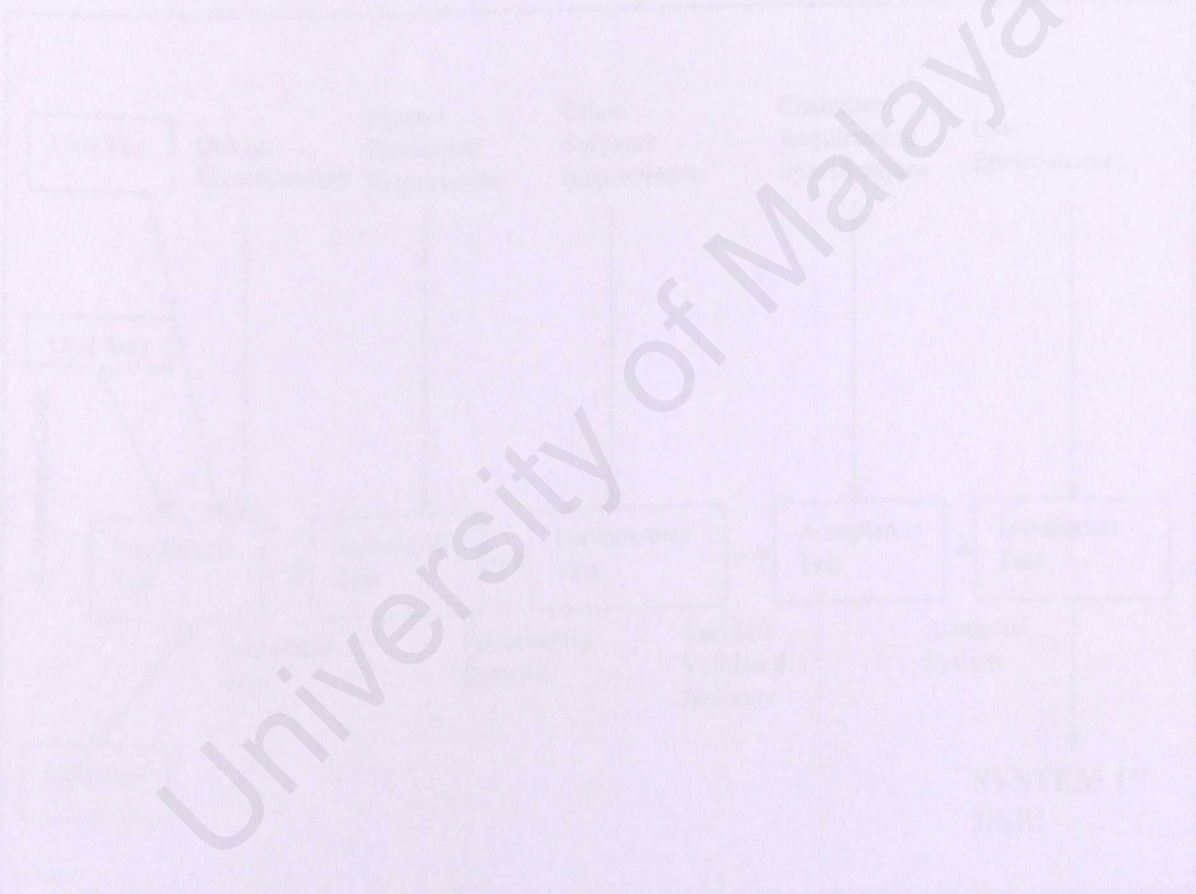


Figure 7-4 System Testing Steps

Unit testing is a type of software testing that focuses on the individual components or units of a software application. It is the first and most basic level of testing, where each part of the program is tested in isolation to ensure it works as intended. This helps to catch errors early in the development process, before they become more complex and difficult to debug.

7.4.2

CHAPTER 7 SYSTEM TESTING

7.1 Introduction

System testing is essential phase to ensure the system performs according to its specifications and in line with user’s requirements and expectations. Testing is done throughout system development not just at the end. The following figure 7-1 is showing the processes in system testing environment.

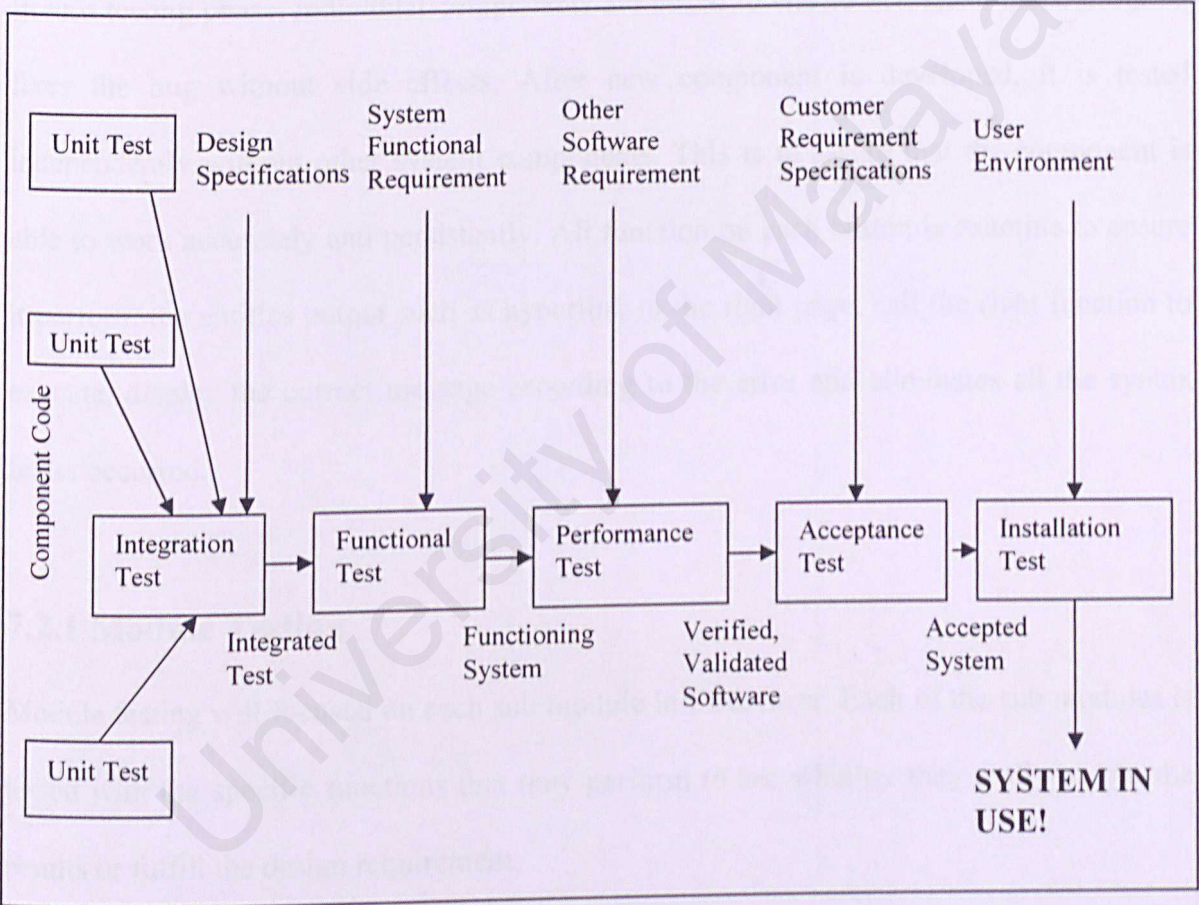


Figure 7-1 System Testing Steps

Testing is preformed to detect the existence of faults and then try to correct it. Therefore, a systematically test procedure is needed to make sure the system is tested thoroughly and completely.

In testing phase, three types of testing have been carried out through FirePower. There are:

- i. Unit and Integration Testing.
- ii. Function Testing
- iii. System Testing

7.2 Unit Testing

In unit testing phase, individual components are tested to ensure that stand-alone program fixes the bug without side effects. After new component is developed, it is tested independently without other system components. This is to assure that the component is able to work accurately and persistently. All function on each button is examine to ensure it perform the entitles output such as hyperlink to the right page, call the right function to execute, display the correct message according to the error and eliminates all the syntax faults occurred.

7.2.1 Module Testing

Module testing will focused on each sub module in FirePower. Each of the sub modules is tested with the specific functions that they perform to see whether they really output the results or fulfill the design requirement.

By using the Process Monitor module as an example, each components or functions in this module has to be examined carefully to discover syntax error or semantic error. Process Monitor module is used to check each application starts with permission from the user in order to prevent various Trojans running in our computers. As a result, every line of the codes related to these functions has to examine one by one to ensure that these functions

fulfilled the user’s requirements. If any errors are discovered, correction and debugging has to be carried out immediately to resolve those problems.

In the unit testing phase, each of the buttons in the appointment module has been examined to make sure that all the buttons are function according to the system environment and requirements. For example, the “No” button will perform all the logical functions such as block the application to start and save the rules in the registry keys.

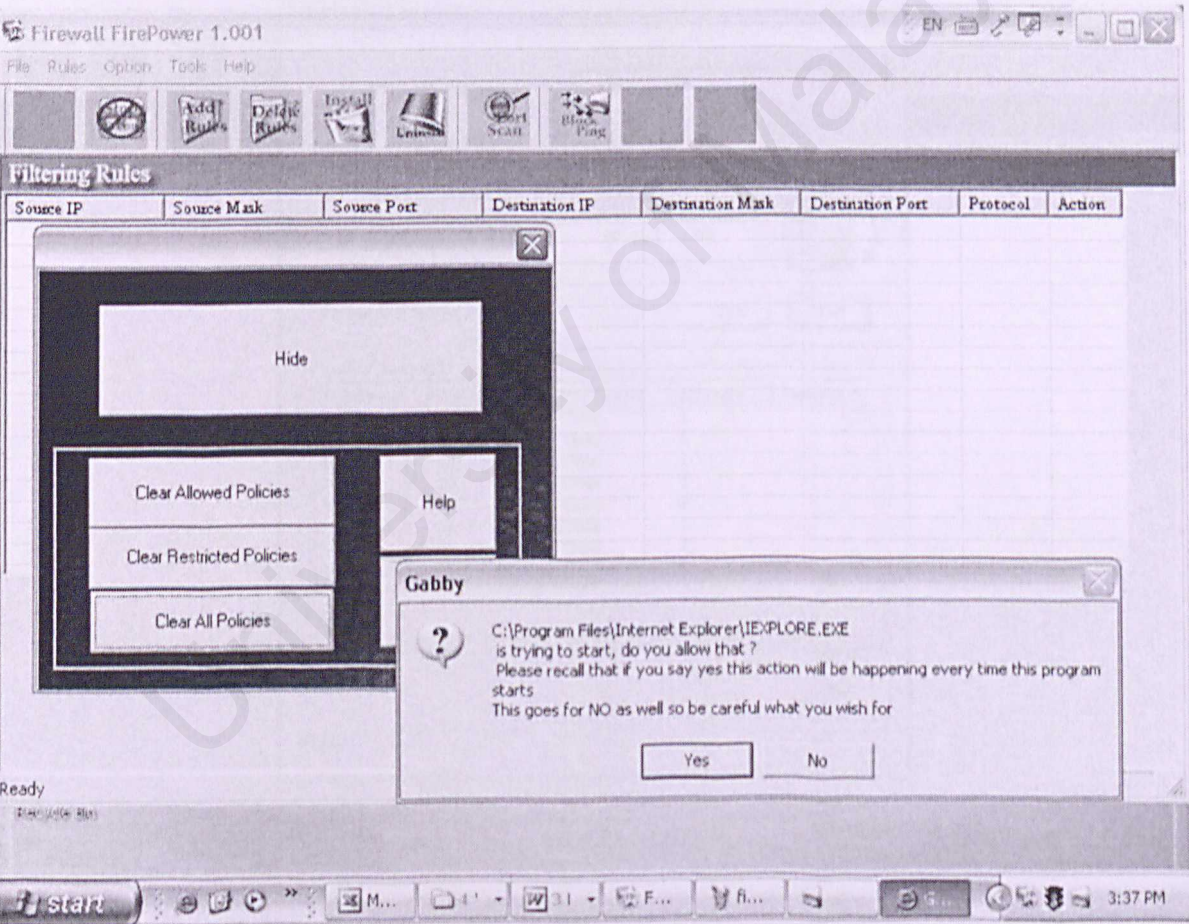


Figure 7-2 Process Monitoring Module

7.2.2 Integration Testing

When the module testing has achieved certain degree of success and meets the objectives, the sub modules are combined into a working system. Integration testing is planned and coordinated so that when a failure occurs, developer has some idea of what caused it. For example, the simple Port Scanner is tested when it integrated into the main system and to make sure it can work properly.

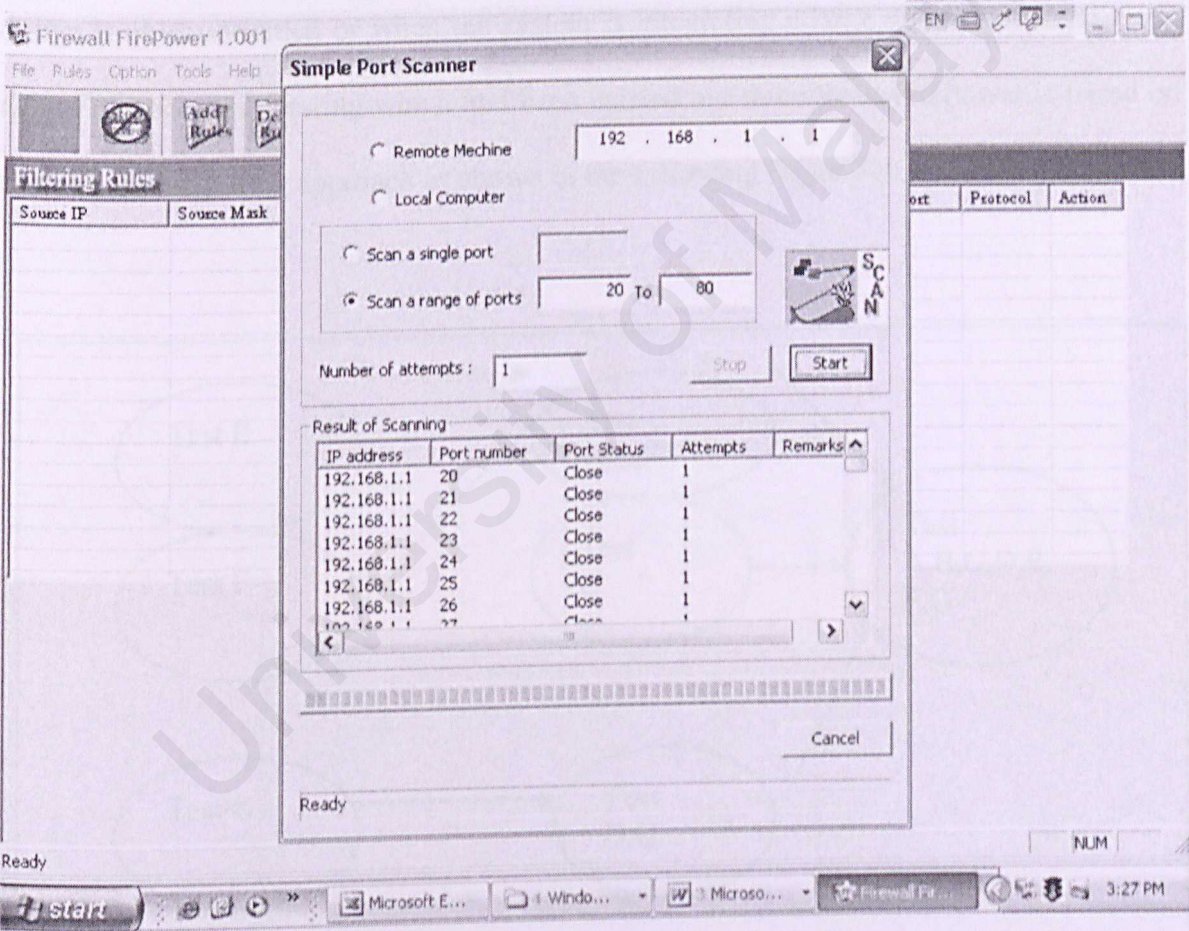


Figure 7-3 Simple Port Scanner Module Integrating Test

7.2.2.1 Bottom - up Integration

Bottom-up testing is one popular approach for merging components to test the larger system. In this method, each component at the lowest level of the system hierarchy is tested individually first. Then the next components to be tested are those that call the previously tested ones. This approach is followed repeatedly until all components are included in the testing. The bottom-up method is useful when many of the low-level components are general-purpose utility routines that are invoked often by others, when the design is object-oriented or when the system is integrating a large number of stand-alone reused components. Testing which had been carried out throughout FirePower is based on Bottom-up Integration approach as shown in the following figure 7-4.

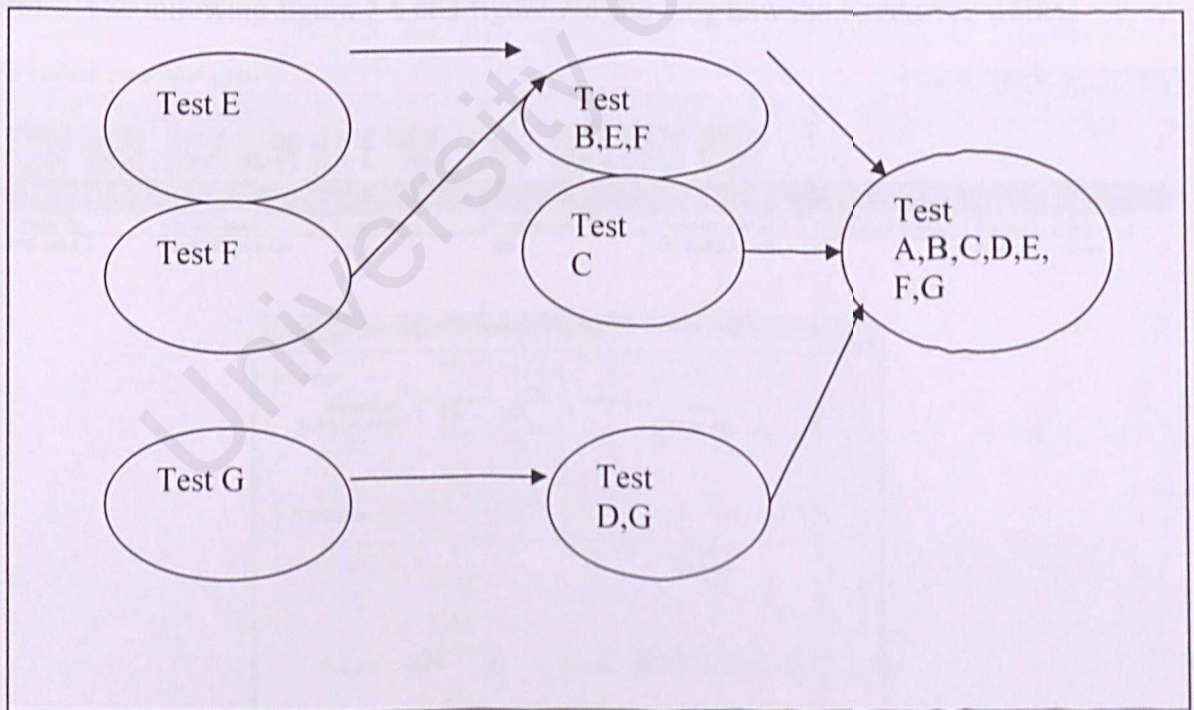


Figure 7-4 Bottom-up Testing

7.3 System Testing

System Testing is differences from unit testing and integration testing. System testing is the ultimate testing procedure. System tests study all the concern issues and behaviors that can only be exposed by testing the entire integrated system or major part of it. The testing process is also concerned with validating the system meets it functional and non-functional requirements. Under system testing, the whole process was simulated and followed through until the end.

Although some of the sub module had been testing for its functionality in the integration testing, now is the really testing to see the integration/interaction from different system. Subsequently, corrections are done to the relevant components upon detection of faults or errors. The following figure 7-5 and figure 7-6 showing how the FirePower works.

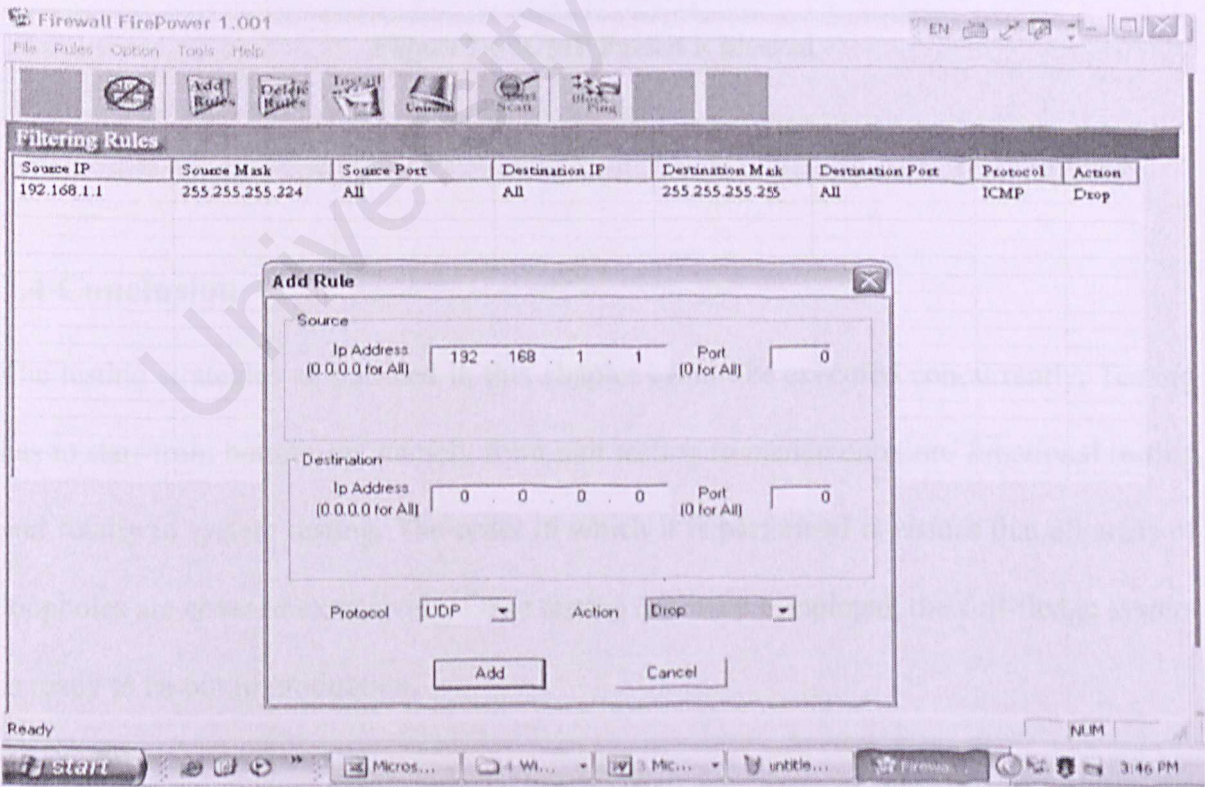
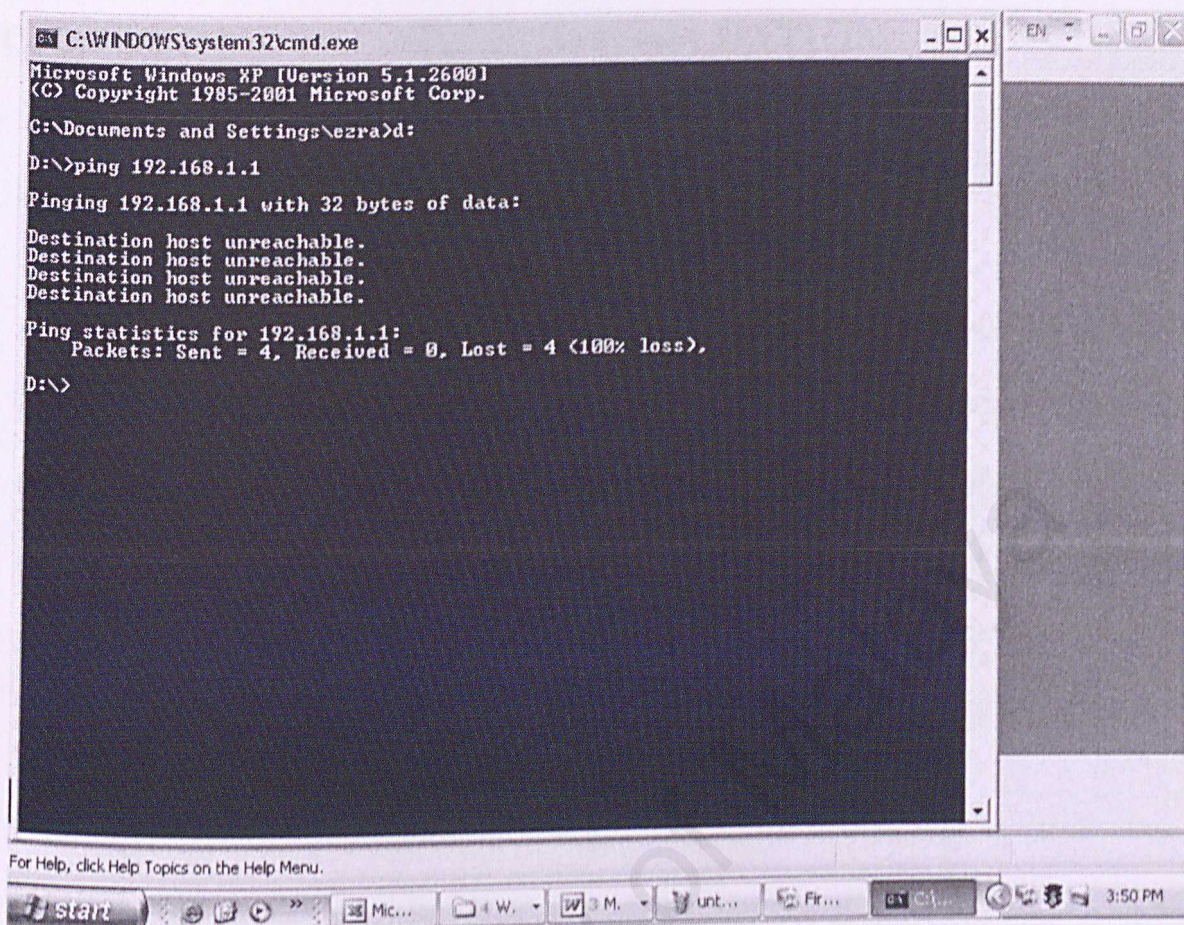


Figure 7-5 Insert Rule to FirePower



```
C:\WINDOWS\system32\cmd.exe
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\ezra>d:
D:\>ping 192.168.1.1

Pinging 192.168.1.1 with 32 bytes of data:
Destination host unreachable.
Destination host unreachable.
Destination host unreachable.
Destination host unreachable.

Ping statistics for 192.168.1.1:
    Packets: Sent = 4, Received = 0, Lost = 4 (100% loss),
D:\>
```

Figure 7-6 ICMP Packet is blocked

7.4 Conclusion

The testing strategies as outlined in this chapter cannot be executed concurrently. Testing has to start from bottom-up, namely from unit testing to implementation/ functional testing, and finally to system testing. The order in which it is performed is ensure that all areas of loopholes are covered extensively. Once testing has been completed, the full-fledge system is ready to be put in production.

CHAPTER 8

SYSTEM EVALUATION

8.1 Introduction

After having gone through the implementation phase, which includes program development and coding, system evaluation is the final phase of developing the system. In this phase, system evaluation involves determine the problems and difficulties which arise during and after the program coding phase, recognizing the system strengths and weaknesses and finally draft out the system limitations and also it future enhancements. This chapter will also present the knowledge and experience gained though the process of developing and implementing FirePower.

These areas will be delved into further details in the following sections.

8.2 System Strengths

As this project is about building a pure computer security firewall application, here is some strength that this program has managed to reveal:

8.2.1 Easy-to-use Application

Ease-of-use is most important aspect in this system. This system specially designed to allow a more efficient and effective protect computer resources from intruders, prevent unauthorized access to the computer. It eliminates as many time-consuming and resource-consuming tasks as possible. On the other hand, this system provides easy to use and user

friendly user interface, no training is needed to be specially conducted to learn how FirePower works. FirePower is extremely easy to use in which all new users will be able to pick in the shortest time as it is equip with graphical button.

8.2.2 Organized and User Friendly Interface

All the modules in FirePower have its own and standardized user interface, no command line is needed and easy to learn. Standardization in user interface design is making FirePower flow easier to understand and reduce the complexity of the system.

8.2.3 System Transparency

System transparency refers to the conditions where users do not have to know or learn about how packet is filtered, how the system structure, how the filter driver is being built and anything related to the system built. Users are just required to know how to communicate and simple configuration with the user interface of the system.

8.2.4 Reliable System with Effective Errors Handling

To avoid run time error, FirePower is developed with error handling. Error message will be displayed when the system encounters exceptions and it will not terminate suddenly. By using “exception handling” block as shown in table below, every error can be handled efficiently. When error occurred, message box will be displayed immediately to the user.

8.3 Problems Encountered and solutions

Several problems have occurred during the development and implementing of FirePower. Some problems have been discovered and solutions have been sought during testing and reference check on the information through Internet Forum and MFC C++ programming reference books. Encountering with these problems has been proven to be valuable experience and guideline in the future.

8.3.1 Difficulties in determining the scope of the application

FirePower involves a lot of network constraints and model. Therefore, basic knowledge is needed as a foundation in building an application of this nature. However, due to lack of knowledge in this field and this field is using latest technology, it is very difficult to determine the scope of the system and how the actual network security firewall works. As a result, a lot of effort is needed to gather information about the scope of the system. Before starting this project plenty of time has been spent in researching the network firewall such as Sygate, Tiny Firewall 6.0 and ZoneAlarm. In this research, we have tried to use the software and learn the way they work.

8.3.2 Understanding on Current System Procedure

From the phase of data gather and analysis, had read some journal, books and tries to understand the current firewall available.

Understanding in details on current firewall application helps in enhances the system. Thus, developers face problem on some details procedure on configure the rules. This is due to the limitations such as:

- Unauthorized access
- Denial of service
- System vulnerable

8.3.3 Difficulties in choosing a development platform, programming language and tools

There are several platforms that we can use to develop a firewall application such as Java, C++ and C# .NET. Besides the platform, choosing a suitable programming language and tools was a critical process as all tools and the programming language have their strengths and weakness. In addition, the availability of the required tools for development is also a major consideration.

Getting information from internet helps in making the decision of choosing platform and development tools. By choosing C++ .NET as the platform have some advantages. C++ .NET is a powerful programming language but equip with easy to use user interface.

8.3.4 Lack of knowledge in develops Firewall

The very fast challenge in developing Firewall such as FirePower is to master and to get familiar with programming language, firewall application behavior and the security features in transferring packets across the network. For instance, to get mastering in using visual studio .NET by using C++ .NET and MFC as the programming language, debugging and compiling file.

By referring to website msdn.microsoft.com, lots of article available there. Besides that lots of valuable information can be gained from codeproject website.

8.4 System Limitations

Owing to the time constraint and the constraint of the programming language itself, there were some limitations in this FirePower. These include:

8.4.1 Lack of functional modules

Although being able to perform the major and important task of observing the traffic, the firewall still lacks certain functionality such as entering more than the number of maximum rules. Well this minor functionality may provide the ideas for the future enhancements of FirePower.

8.4.2 Lack of Network Utilities Display

The FirePower is not able to show the traffic bandwidths usage.

8.5 Future Enhancements

Some functionality of the system can be enhanced in order to improve the quality of the system. The following are some recommendations be developer and user on how to enhance the system that developed.

8.5.1 Increase the number of rules

In order to guard the traffic effectively, more rules is needed. Beside that more powerful filtering driver been developed.

8.5.2 Module Enhancement

More effective application filtering is developed.

8.6 Knowledge and experienced gained

Through out the process in developing FirePower, a lot of knowledge and experience can be gained. First of all, mastering in C++ .NET programming language, which are the powerful programming languages nowadays. Besides that though out this project I have the chance to learn the latest technology and networking concept such as packet filtering concept. Knowledge can be gained through self-study, information from internet, practically applying the knowledge and getting advice from others.

More opportunities in exposing to PC environment including setting up tools, configuring hardware and software had eventually helps in experiencing the real and practical techniques in performing those mentioned tasks.

In developing FirePower, an opportunity given to get involve in designing systems, system tray , programs and interfaces of the system.

Presenting proposed FirePower and public speaking had also helps in improving presentation skill. In spite of this, experiences are gained from such an activity and to improve one to become more self-confidence when facing the public.

REFERENCES

In addition, developer has to be independent enough when performing task in developing system. Developer also must learn to be a problem solver, patient, critical and an analytical thinking to resolve problems and to face challenges.

8.7 Conclusion

After implementation, FirePower should be evaluated. Problems that encountered were analyzed and the appropriate solutions were taken carefully. Overall, FirePower strengths fulfill the functional and non-functional requirements as planned at the start of this project. Although there are some constraints or limitations in FirePower but there can be future enhancement. Knowledge and experience gained through this project also being evaluated.

REFERENCES

Firewalls 24Seven, Second Edition

by Matthew Strebe and Charles Perkins

Sybex © 2002 How to design, implement, and maintain a secure network.

Deitel 1997. C++ How to Program. New Jersey: Prentice Hall

Deitel 1997. Java How to Program. New Jersey: Prentice Hall

M.Melly.1995.Object Oriented Basic Concepts and Advantages[online]. Availale
from: www.mmrg.ecs.soton.ac.uk/publications/archieve/melly1995a/html/node3.html

Programming with Microsoft Visual C++ .NET, Sixth Edition

by George Shepherd and David Kruglinski

Microsoft Press © 2003

The in-depth reference that covers both classic, core Windows competencies and
modern .NET programming.

<http://www.codeproject.com>

<http://ietf.org>

<http://compnetworking.about.com/>

<http://www.c-sharpcorner.com/Networking.asp>

<http://www.hackingexposed.com/tools/tools.html>

<http://www.uml原因.edu.my>

<http://www.cprogramming.com/begin.html>

<http://www.ibiblio.org/javafaq/javatutorial.html>

<http://www.planet-source-code.com>

<http://www.sysinternals.com>



Figure 10.1

APPENDIXES

Outline

- A. Installation
- B. System Requirement
- C. Getting Started

A. Installation

1. Open folder Installer under folder FirePower Ver 1.001 and double click FirePower Ver. 1.001 Windows Package Installer. (Make sure you read the user manual before installing this firewall application.)
2. Then a setup wizard will prompt up and read carefully the instruction (Figure 10.1).
After that, proceed to the next steps.

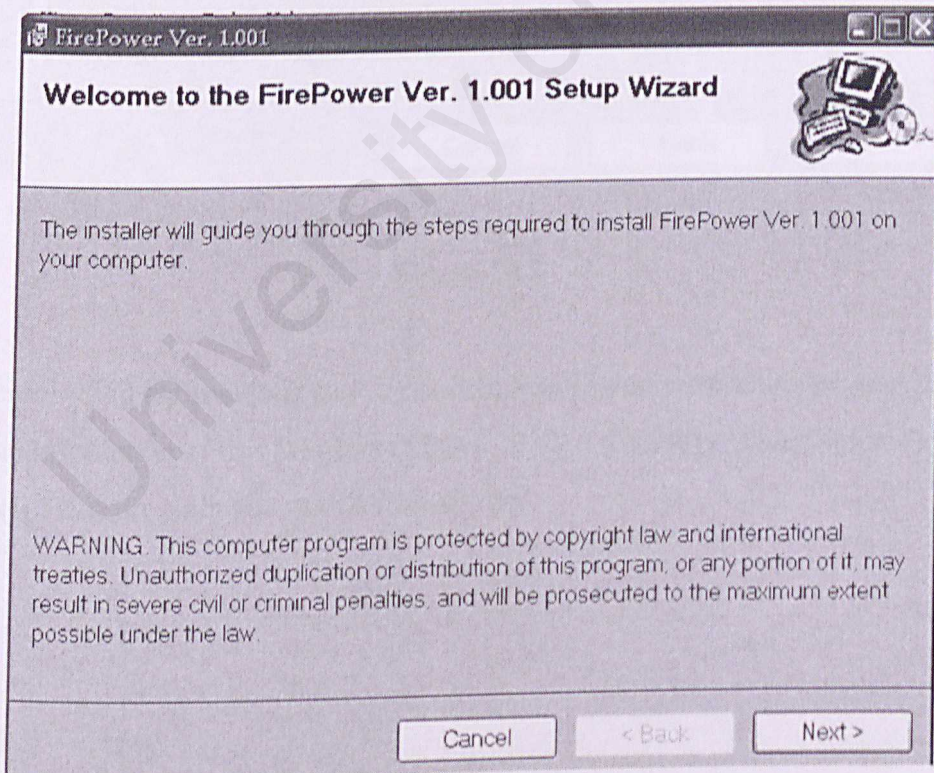


Figure 10.1

3. Then, a wizard that require to select the installation folder for FirePower Ver. 1.001 as shown in figure 10.2. Follow the instruction and proceed to the next steps. (Disk Cost

will show out the space needed by this program to install on your hard disk. You can click Browse button to change the location of installation files.)

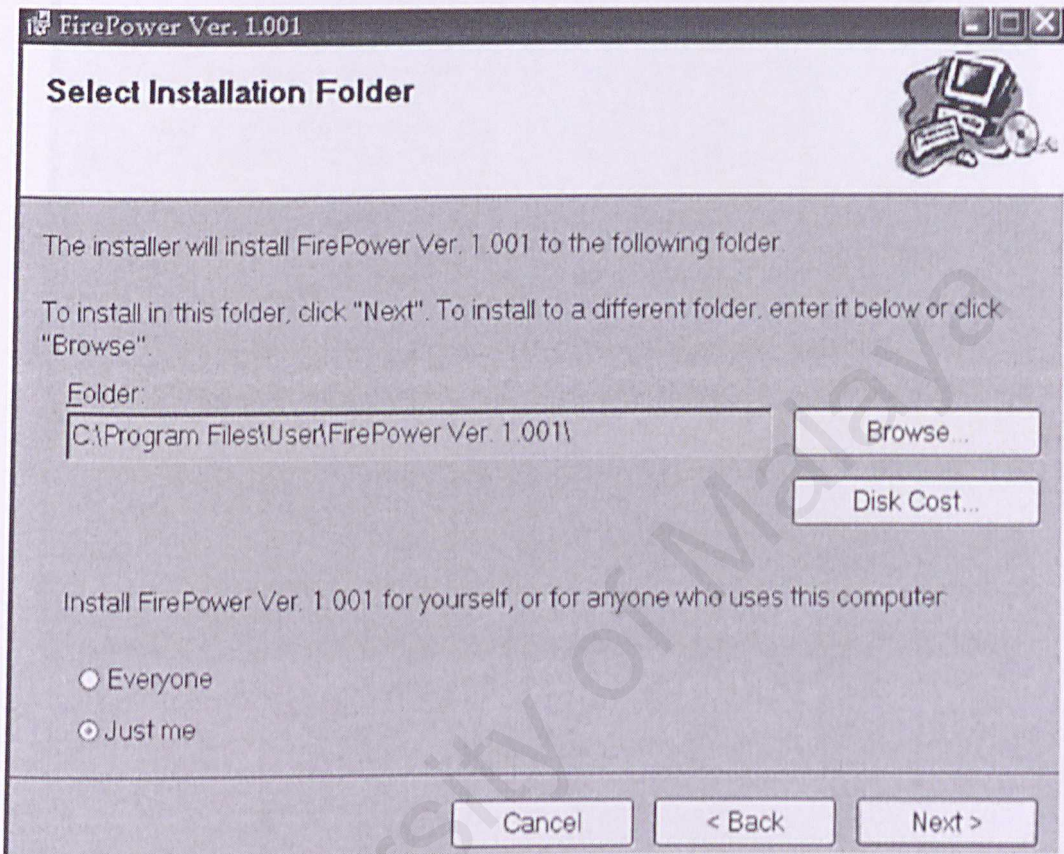


Figure 10.2

4. After proceeding the previous step, a confirm installation wizard will prompt out to confirm the installation of this program (Figure 10.3). Click Next to confirm and the installation of this program will start automatically.

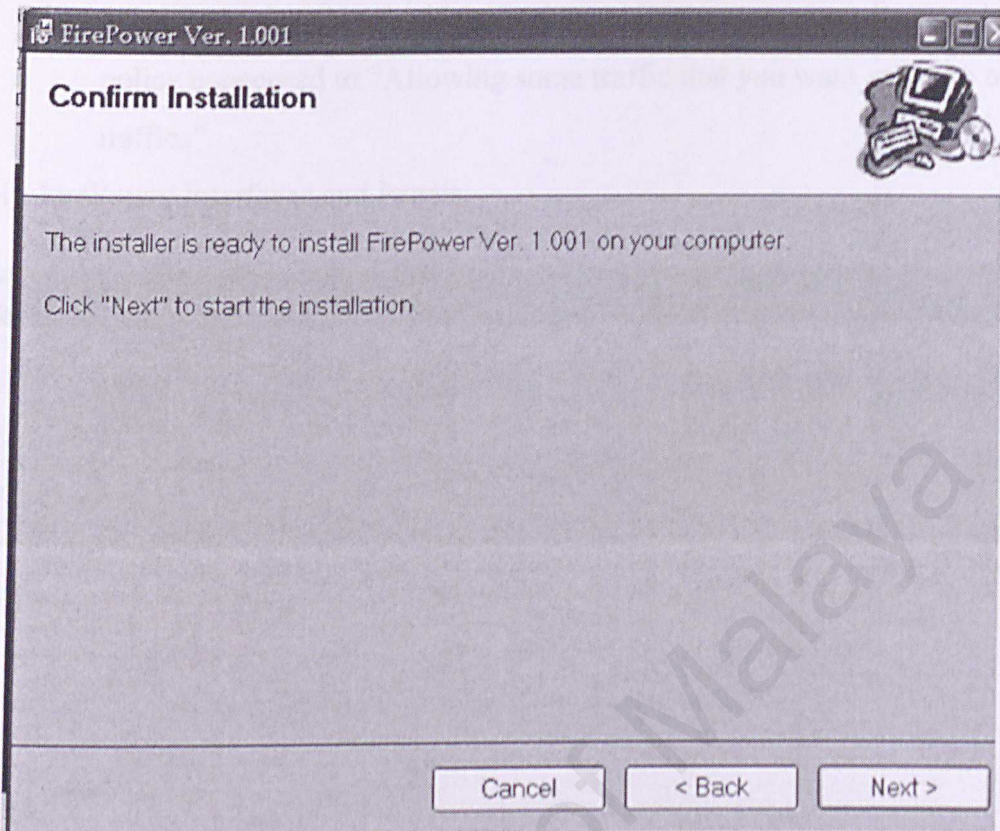


Figure 10.3

B. System Requirement

Processor: Pentium II 300 MHZ and above.

All Windows Operating System:

Windows 98, Windows 2000, Windows ME, Windows 2000 Server, Windows 2003 Server, Windows NT version 4, and Windows XP.

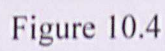
Space needed: 24 MB

C. Getting Started

I. Introduction

FirePower Ver. 1.001 is developed as a packet-filtering based firewall application which more suitable for small and simple network or personal using on personal computer. This firewall is using FireHook Driver (Refer to Guide.doc under FirePower folder.). For implementing this firewall

II. FirePower Interfaces and Panels



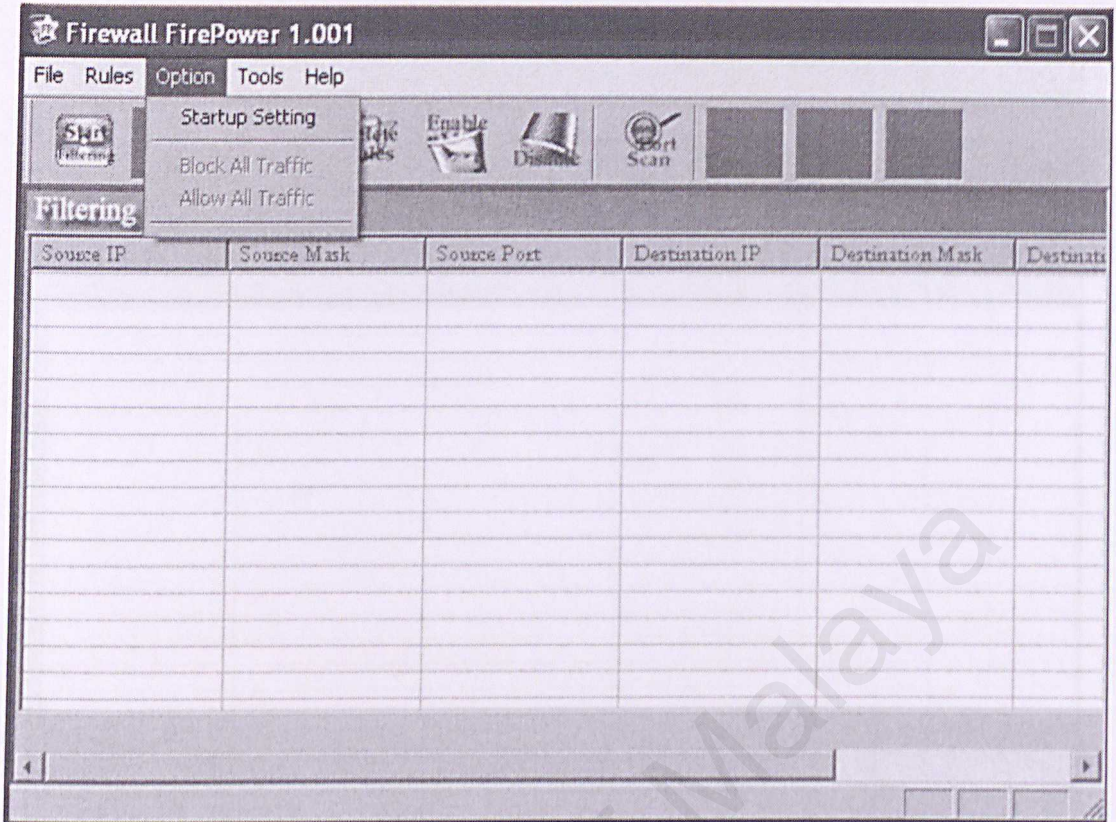


Figure 10.7

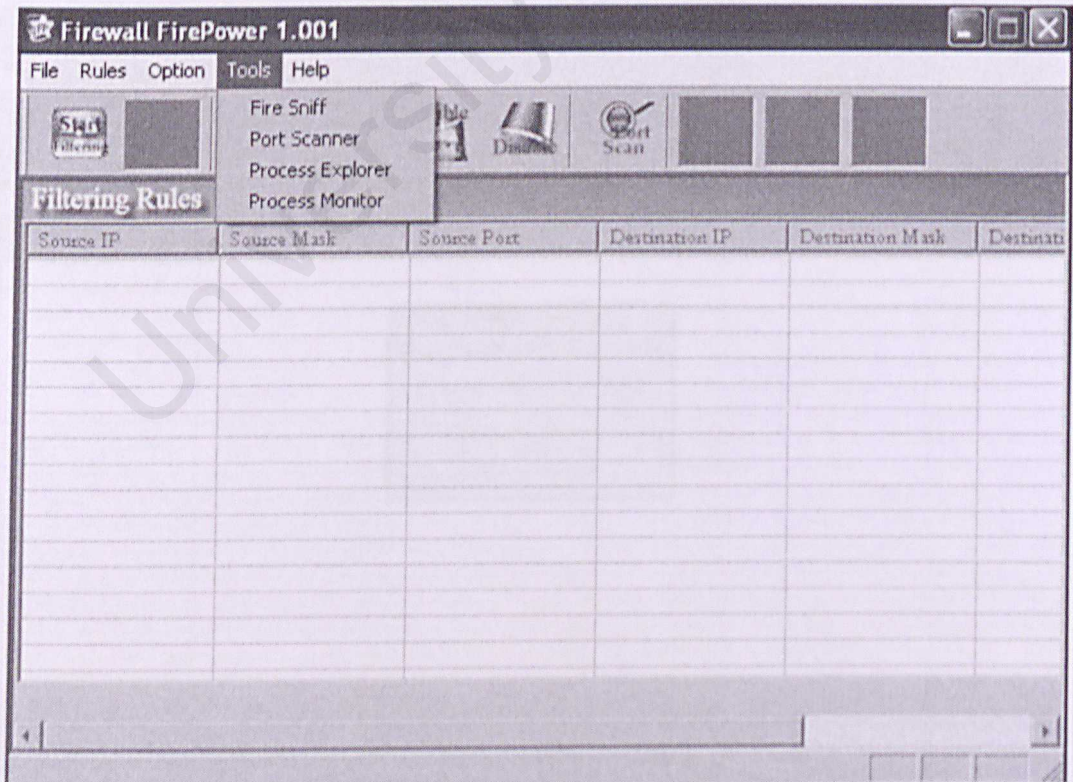


Figure 10.8

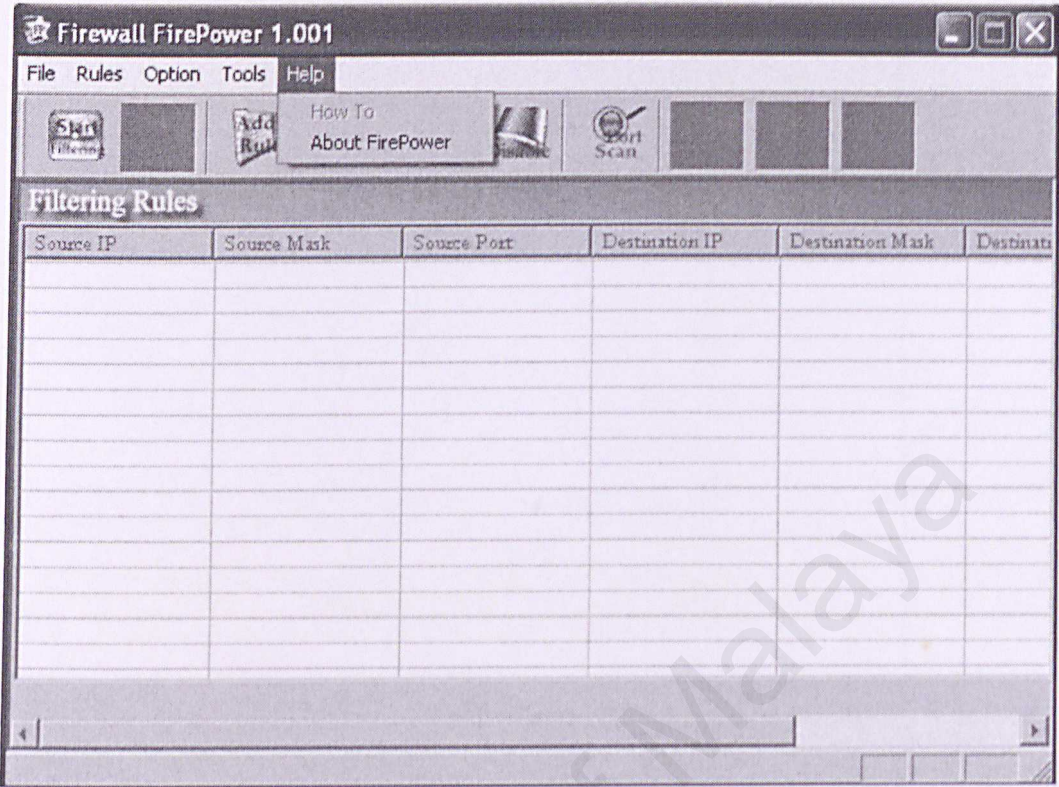


Figure 10.9

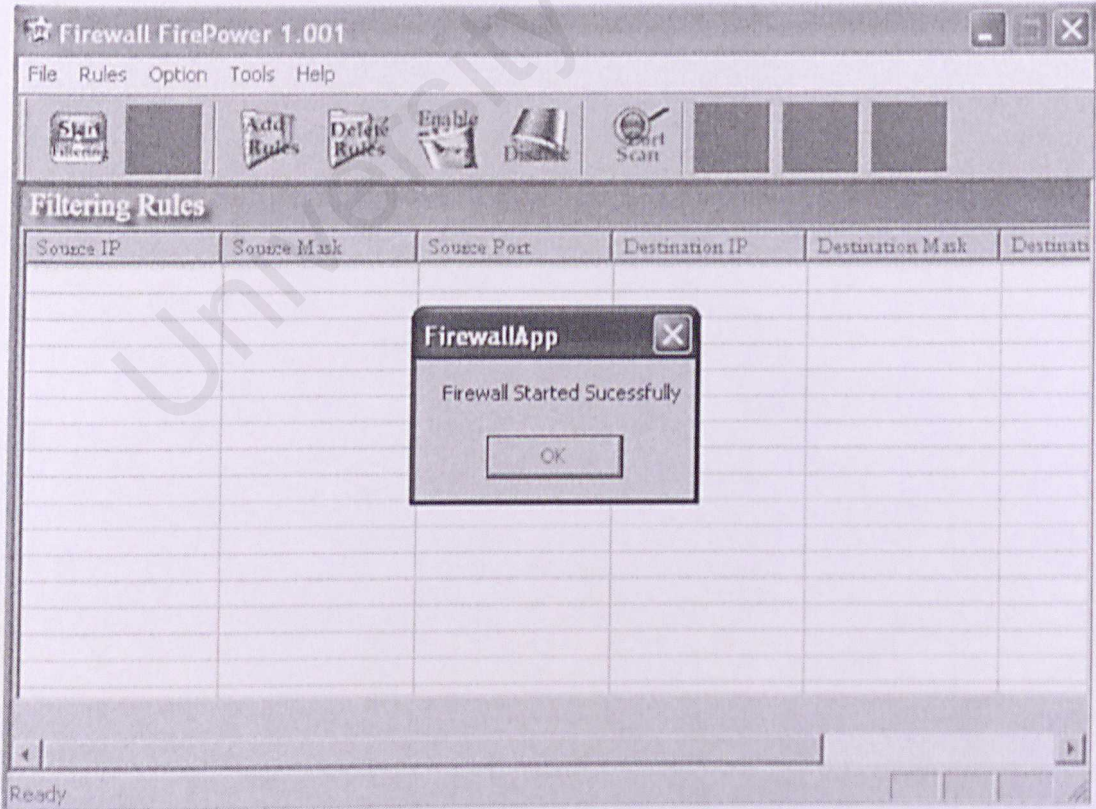


Figure 10.10