THE NETFILTER/IPTABLES FRAMEWORK IN LINUX 2.4.X

Supervised by:- Mr. Nazim-ud-din
Deputy Director
Computer Centre

Submitted by:- Ghazanfar Ali Khan
PGD 10

Presented to the Faculty of the Computer Center
Quaid-i-Azam University, Islamabad
in partial fulfillment of the requirements for the degree of
Post Graduate Diploma in Computer science

2002

THE NETFILTER/IPTABLES FRAMEWORK IN LINUX 2.4.X

Supervised by:- Mr. Nazim-ud-din
Deputy Director
Computer Centre

Submitted by:- Ghazanfar Ali Khan
PGD 10

Presented to the Faculty of the Computer Center
Quaid-i-Azam University, Islamabad
in partial fulfillment of the requirements for the degree of
Post Graduate Diploma in Computer science

2002

Quaid-i-Azam University, Islamabad
Computer Center

This is certified that we have read the project report submitted by **Ghazanfar Ali Khan** and it is our judgment that the report is of sufficient standard to warrant its acceptance by the Quaid-i-Azam University, Islamabad, for the Post Graduate Diploma in Computer Science.

COMMITTEE

1. External Examiner

_____

2. Supervisor

Mr. Nazim-ud-din (Deputy Director)
Computer Center
Quaid-e-Azam, University
Islamabad

_____

3. Director

Dr. Ghulam Mohammad
Computer Center
Quaid-e-Azam, University
Islamabad

_____

Acknowledgements

The whole praise to ALLAH, the beneficent, the merciful who endowed in me the capabilities to complete this academic effort. His blessings showered on me and I am unable to thank him.

In this project my supervisor, Mr. Nazim-ud-Din advised and encouraged me to his maximum and I am particularly indebted to his guidance and help.

My regards are for all my teachers, especially Mr. Tiwana whose constant persuasions and encouragement kept up my enthusiasm to complete this study. Special thanks to my teacher, Mr Mohammad Sher, of Data communication & Networks for providing the best understanding of internals and working of TCP/IP stack.

I am grateful to all my class fellows for their valuable companionship, encouragement, moral support, and co-operation. I would like to acknowledge moral support and best wishes of my friends. Credits go for my brother, Mohammad Ali Khan, for first time introducing me to the Linux and the benefits of the Open Source. He also helped me in the formatting & editing.

I would be failing in my duty if I don't acknowledge my gratitude to my teacher Dr. Ghulam Mohammad, who has always been a source of inspiration for me.

And at the last but not least, I would like to admit that I owe all my achievements to my loving mother, who mean most to me, and whose prayers are a source of determination for me.

Ghazanfar Ali Khan
January, 2003
Quaid-i-Azam, University
Islamabad

To my loving Family

And

Friends

Abstract

The Internet is a worldwide collection of networks that all use a common protocol for communications. Many organizations are in the process of connecting to the Internet to take advantage of Internet services and resources. Businesses and agencies are now using the Internet or considering Internet access for a variety of purposes, including exchanging e-mail, distributing agency information to the public, and conducting research. Many organizations are connecting their existing internal local area networks to the Internet so that local area network workstations can have direct access to Internet services.
Internet connectivity can offer enormous advantages, however security needs to be a major consideration when planning an Internet connection. There are significant security risks associated with the Internet that often are not obvious to new (and existing) users. In particular, intruder activities as well as vulnerabilities that could assist intruder activity are widespread. Intruder activity is difficult to predict and at times can be difficult to discover and correct.

"It is easy to run a secure computer system. You merely have to disconnect all dial-up Connections and permit only direct-wired terminals, put the machine and its terminals in a shielded room, and post a guard at the door."

—F.T. GRAMPP AND R.H. MORRIS

For better or for worse, most computer systems are not run that way today. Security is, in general, a trade-off with convenience, and most people are not willing to forgo the convenience of remote access via networks to their computers. Inevitably, they suffer from some loss of security. It is my purpose here to discuss how to minimize the extent of that loss.

The situation is even worse for computers hooked up to some sort of network. Networks are risky for at least three major reasons. First, and most obvious, more points now exist from which an attack can be launched. A second reason is that you have extended the physical perimeter of your computer system. In a simple computer, everything is within one box. The CPU can fetch authentication data from memory, secure in the knowledge that no enemy can tamper with it or spy on it. Traditional mechanisms—mode bits, memory protection, and the like—can safeguard critical areas. This is not the case in a network. Messages received may be of uncertain provenance; messages sent are often exposed to all other systems on the net. Clearly, more caution is needed. The third reason is subtler, and deals with an essential distinction between an ordinary dial-up modem and a network. Modems, in general, offer one service, typically the ability to log in.

Networked computers offer many services: *login*, file transfer, disk access, remote execution, phone book, system status, etc. Thus, more points are in need of protection—points that are more complex and more difficult to protect. A networked file system, for example, cannot rely on a typed password for every transaction. Furthermore, many of these services were developed under the assumption that the extent of the network was comparatively limited. In an era of connectivity, that assumption has broken down, sometimes with severe consequences. Networked computers have another peculiarity worth noting: they are generally not singular entities. That is, it is comparatively uncommon, in today's environment, to attach a computer to a network solely to talk to "strange" computers. More commonly, organizations own a number of computers, and these are connected to each other and to the outside world. This is both a curse and a blessing: a curse, because networked computers often need to trust their peers, to some extent, and a blessing, because the network may be configurable so that only one computer needs to talk to the outside world. Such dedicated computers, often called "firewall gateways," are the topic of my study. My purpose here is twofold. First, I wish to show that such gateways are necessary, and that there is a real threat to be dealt with. Second, I wish to show that this strategy is useful. That is, a firewall, if properly deployed against the expected threats, will provide an organization with greatly increased security.

Background

The Internet is a vital and growing network that is changing the way many organizations and individuals communicate and do business. However, the Internet suffers from significant and widespread security problems. Intruders have been observed to target specific sites for intrusions by methodically scanning host systems for vulnerabilities. Intruders often use automated probes, i.e., software that scans all host systems connected to a site's network. This is sometimes referred to as probing a site. Many agencies and organizations have been attacked or probed by intruders, with resultant high losses to productivity and reputation. In some cases, organizations have had to disconnect from the Internet temporarily, and have invested significant resources in correcting problems with system and network configuration. Sites that are unaware of or ignorant of these problems face a significant risk that network intruders will attack them. Even sites that do observe good security practices face problems with new vulnerabilities in networking software and the persistence of some intruders.

A number of factors have contributed to this state of affairs. The fundamental problem may be that the Internet was not designed to be very secure, i.e., open access for the purposes of research was the prime consideration at the time the Internet was implemented. However, the phenomenal success of the Internet in combination with the introduction of different types of users, including unethical users, has aggravated existing security deficiencies to the extent that wide-open Internet sites risk inevitable break-ins and resultant damages. Other factors include the following:

- **Vulnerable TCP/IP services** - a number of the TCP/IP services are not secure and can be compromised by knowledgeable intruders; services used in the local

area networking environment for improving network management are especially vulnerable.

- **Ease of spying and spoofing** - the majority of Internet traffic is un-encrypted; e-mail, passwords, and file transfers can be monitored and captured using readily available software. intruders can then re-use passwords to break into systems,
- **Lack of policy** - many sites are configured unintentionally for wide-open Internet access without regard for the potential for abuse from the Internet; many sites permit more TCP/IP services than they require for their operations and do not attempt to limit access to information about their computers that could prove valuable to intruders, and
- **Complexity of configuration** - host security access controls are often complex to configure and monitor; controls that are accidentally miss-configured often result in unauthorized access.

Solutions

Fortunately, there are readily available solutions that can be used to improve site security. A firewall system is one technique that has proven highly effective for improving the overall level of site security. A firewall system is a collection of systems, routers, and policy placed at a site's central connection to a network. A firewall forces all network connections to pass through the gateway where they can be examined and evaluated, and provides other services such as advanced authentication measures to replace simple passwords. The firewall may then restrict access to or from selected systems, or block certain TCP/IP services, or provide other security features.

A simple network usage policy that can be implemented by a firewall system is to provide access from internal to external systems, but little or no access from external to internal systems. However, a firewall does not negate the need for stronger system security. There are many tools available for system administrators to enhance system security and provide additional logging capability. Such tools can check for strong passwords, log connection information, detect changes in system files, and provide other features that will help administrators detect signs of intruders and break-ins.

Purpose

The purpose of this study is to provide a basis of understanding of how firewall work and the steps necessary for implementing firewalls using Netfilter/Iptables framework within the Linux 2.4 kernel.

Ghazanfar Ali Khan.
ghazanfar@linuxmail.org

Project Brief

Project Title	The Netfilter/Iptables Framework in Linux 2.4.x
Organization	Computer Centre Quaid-I-Azam University, Islamabad
Undertaken By	Ghanzafar Ali Khan
Supervised By	Mr.Nazim-ud-Din Deputy Director Computer Centre Quaid-I-Azam University Islamabad
Starting Month	July, 2002
Completion Month	January, 2003
Software Used	Iptables 1.9
Operating System	Redhat Linux 7.3 Kernel 2.4.18

Table Of Contents

Chapter 1 Network Security – Overview
1.1 Local & Wide Area Networks....1
1.2 The Internet....1
1.3 Other Network Types....2
1.3.1 Enterprise-Wide Networks....2
1.3.2 Virtual Private Networks....3
1.3.3 Intranets....3
1.4 Modem Connections....3
1.5 The OSI Model for Network Architecture....4
1.6 The Security Threat....4
1.7 An Internet Policy....5
1.7.1 The Firewall....5

Chapter 2 The Internet and Internet Security
2.1 Common Services....6
2.1.2Internet Hosts....7
2.2 Overview of TCP/IP Internals....8
2.2.1IP....8
2.2.2TCP....9
2.2.3UDP....9
2.2.4 ICMP....10
2.2.5 TCP and UDP Port Structure....10
2.3 Security-Related Problems....11
2.3.1 Weak Authentication....12
2.3.2 Ease of Spying/Monitoring....12
2.3.4 Ease of Spoofing....13
2.3.5 Source Routing....14
2.3.6 ICMP Redirects & Redirect Bombs....14
2.3.7 Denial of Service....14
2.3.8 Flawed LAN Services and Mutually Trusting Hosts....15
2.3.9 Complex Configuration and Controls....15
2.3.10 Host-based Security Does Not Scale....16

Chapter 3 Introduction to Firewall
3.1 The Firewall Concept....17
3.2 Why Firewalls?....18
3.2.1 Protection from Vulnerable Services....18
3.2.2 Controlled Access to Site Systems....18

3.2.3 Concentrated Security....19
3.2.4 Enhanced Privacy....19
3.2.5 Logging and Statistics on Network Use, Misuse....19
3.2.6 Policy Enforcement....19
3.3 Issues and Problems with Firewalls....20
3.3.1 Restricted Access to Desirable Services....20
3.3.2 Large Potential for Back Doors....20
3.3.3 Little Protection from Insider Attacks....20
3.3.4 Other Issues....21
3.4 Firewall Components....21
3.4.1 Network Policy....21
3.4.1.1 Service Access Policy....22
3.4.1.2 Firewall Design Policy....22
3.4.2 Advanced Authentication....23
3.4.3 Packet Filtering....23
3.4.3.1 Which Protocols to Filter....24
3.4.4 Application Gateways....25
3.4.5Circuit-Levelgateways....27
3.5 Firewall Architecture....27
3.5.1 Packet Filtering Firewall....27
3.5.2 Dual-homed Gateway Firewall....27
3.5.3 Screened Host Firewall....28
3.5.4 Screened Subnet Firewall....30
3.6 Integrating Modem Pools with Firewalls....31

Chapter 4 The Netfilter framework in Linux 2.4
4.1 Netfilter basics / concepts....34
4.2 What is netfilter?....34
4.3 Why did we need netfilter?....35
4.4 Netfilter architecture in IPv4....35
4.4.1 Netfilter base....36
4.4.2 Packet selection: IP tables....36

Chapter 5 Packet filtering using iptables
5.1 Traversing of tables and chains....38
5.2 Mangle table....42
5.3 Nat table....42
5.4 Filter table....43

Chapter 6 The state machine
6.1
Introduction....44
6.2 The conntrack entries....45
6.3 Userland states....46
6.4 TCP connections....47
6.5 UDP connections....49

6.6 ICMP connections....50
6.7 Default connections....52

Chapter 7 How a rule is built
7.1 Basics....53
7.2 Commands....54
7.3 Matches....56
7.3.1 Generic matches....56
7.3.2 Implicit matches....58
7.3.3 Explicit matches....60
7.4 Targets/Jumps....66
7.4.1 ACCEPT target....66
7.4.2 DROP target....67
7.4.3 QUEUE target....67
7.4.4 RETURN target....67
7.4.5 LOG target....67
7.4.6 MARK target....67
7.4.7 REJECT target....67
7.4.8 TOS target....68
7.4.9 MIRROR target....68
7.4.9 SNAT target....68
7.4.10 DNAT target....69
7.4.11 MASQUERADE target....70
7.4.12 REDIRECT target....71
7.4.13 TTL target....71
7.4.14 ULOG target....71

Chapter 8 Example Implementation at Computer Center Q.A.U
8.1 System Selection....74
8.1.1 Hardware Requirements....74
8.1.2 Software requirements....74
8.2 Preparing the Linux system....74
8.3 Configuring two network cards....75
8.4 Testing the network....76
8.5 Initial System security....77
8.5.1 BIOS/CMOS Settings....77
8.5.2 /tmp....77
8.5.3 Control-Alt-Delete keyboard shutdown command....77
8.5.4 Disabling the ability to run INIT in interactive mode....77
8.5.5 Change what system daemons get loaded by editing the following files in "/etc/rc.d/"....77
8.5.6 Shutting down most of inetd / xinetd....78
8.5.7 Shadow Passwords....78
8.5.8 Disabling miscellaneous cron stuff....78
8.6 Firewalling Script....79
8.6.1 Configuration options....79

8.6.2 proc set up.....79
8.6.3 Displacement of rules to different chains.....80
8.6.4 Setting up default policies.....82
8.6.5 Setting up user specified chains in the filter table.....82
8.6.5.1 The bad_tcp_packets chain.....83
8.6.5.2 The allowed chain.....83
8.6.5.3 The TCP chain.....83
8.6.5.4 The UDP chain.....84
8.6.5.5 The ICMP chain.....84
8.6.5.6 INPUT chain.....84
8.6.5.7 FORWARD chain.....85
8.6.5.8 OUTPUT chain.....86
8.6.5.9 PREROUTING chain of the nat table.....86
8.6.5.10 Starting SNAT and the POSTROUTING chain.....86
8.7 Script.....87

Chapter 1

Network Security – Overview

The increasing use of network services such as the Internet has enhanced awareness of the need for security in distributed computer systems. Whenever information is transmitted it becomes vulnerable because:

- It is no longer protected inside a closed, secure operating system
- It is subject to the faults and failure of equipment owned and maintained by others
- It is subject to potential attack by any person with a network connection.

In addition the Internet itself has a subversive element within it:

- There are groups devoted to the subject of "cracking", where information on soft targets can be found.
- Software and instructions, which together form "system-penetration kits", are available, free, on the Internet.

This section addresses the problems associated with computer networks.

1.1 Local & Wide Area Networks

Computer networks, which occupy a small geographical area, such as a single building or a campus site, are referred to as *local area networks* (LANs). A terminal on a LAN typically broadcasts a message to the whole network and the terminal, which is the intended recipient, accepts the message, while the other connected terminals ignore it. These small networks might have any number of users ranging from two to a few hundred. The data moves between terminals as a fast serial stream, usually using the packet-switching Ethernet standard, which is capable of speeds of up to 10/100 mega-bits per second.

A *wide area network* (WAN) connects together terminals, which are geographically separate. Governments connect their military establishments using WANs, as do private companies, financial institutions, and international companies. WAN systems comprise a variety of terrestrial and satellite communication systems, the majority of which are leased and can be considered as "private" to the WAN owner. Information transfer is usually by packet-switching technology.

1.2 The Internet

The Internet owes its existence to the work undertaken by the Defense Advanced Research Projects Agency in the 1970s. At this time the extensive military-funded WAN, known as ARPANET, was thought to be highly vulnerable to nuclear attack, and it was reconstructed into a great number of smaller WANs. A new method was devised for transmitting data across these smaller networks so that, in the event of localized

destruction, the information would automatically seek an alternative route to its destination. To make this work it was necessary to allow systems on different networks to communicate, and two new protocols, the *Transmission Control Protocol* (TCP) and the *Internet Protocol* (IP), (collectively known as TCP/IP), were defined.

Universities, computer manufacturers and other technical non-military site hade been allowed to connect to ARPANET and the ease of connection afforded by the TCP/IP led to rapid growth in the early 1980s. In 1984, the classified portion of the WAN was split away to become what is now known as the Defense Data Network (DDN), and the remainder of the WAN became known as The Internet.

No institution owns, controls or regulates the Internet. The information on it does not travel along predictable routes, it is often slower than a dedicated corporate-owned WAN, but, thanks to the technology derived from the military, the probability that the information will reach its destination is very high.

The World Wide Web or **WWW** is a service within the Internet, offering a common look and feel to the global information on the Web. The Web typically offers the user a "point-and-click" environment, where web sites (Internet destinations) can be visited sequentially in a seamless and effortless manner. The success of the Web has been phenomenal and it has overtaken the **email** service to become the leading light of the Internet.

Businesses are using the Web to search for information, retrieve product information including software, and to collaborate with partners. They may also want to set up their own web site for advertising/marketing, or to give technical support and product-related information to customers. When security issues are further resolved, the web may also be used extensively for electronic commerce.

1.3 Other Network Types

The success of the Internet has prompted various suggestions for other network scenarios, which are being actively promoted by users and vendors alike. In this fast-evolving arena, labels such as *Enterprise-Wide Network*, *Virtual Private Network* and *Intranet* are being used to describe aspects of one single trend - the increase in *connectivity*.

1.3.1 Enterprise-Wide Networks

When an organization begins connecting its computers together, it is, by design or by default, heading towards what in computing terms is called a distributed system. Connectivity continues until a *Enterprise-Wide Network* emerges. This network will probably include dial-in access (for employees with portable computers) and may also include the sites and systems of collaborating companies and contractors. There will invariably be a connection to the Internet.

Many companies, anxious to maintain a security architecture on this evolving distributed system, divide the network into *trust domains*, where the level of security is known, and audit control can be maintained. New domains are judged to be trusted or untrusted, and

if they are untrusted, they are connected to the existing network by a *firewall*. At any one time the company may not know the exact location of its electronic perimeter wall, but it achieves the necessary overall high level of security by maintaining its trusted domains and by the judicious use of firewalls.

1.3.2 Virtual Private Networks

Advocates of the Virtual Private Network (VPN) propose that encryption be applied to the packets of data that are transmitted over public lines, including any lines which are also part of the existing internet. With the data thus uncompromised, a company could operate a Wide Area Network without the need for expensive, dedicated lines, and maintain private data communication over distances that far exceeded even the most ambitious WAN. However, unlike a protected WAN, the lines could be subjected to traffic analysis and network flooding.

1.3.3 Intranets

The Intranet allows greater connectivity yet. The idea is simple- the multimedia attraction and ease-of-use of the WWW can be adopted in-house by companies who wish to encourage better communication between staff, between departments and between levels of management. Thus, all employees would be on the *intranet*, a low-cost internal web, sharing vast amounts of corporate information and resources. Staff would benefit from high connectivity (to each other) on a net which would be intrinsically secure, since *no connection to the Internet* need be made at all. Being an internal web, encryption techniques would be simple to implement and would provide company-wide protection against commercial espionage.

Advocates of the *Intranet* envisage that an enterprise might wish to progress, through modular extensions of connectivity, to full Internet access for its workforce and to virtual private networking using inexpensive lines. That would certainly be possible, and might even be desirable, provided that the security implications were fully understood. The security advantage of the *Intranet* is that it *does not* connect to the outside world.

1.4 Modem Connections

Perhaps the simplest and least expensive method of communication is where the computer answers the telephone or where it initiates a telephone call.

Incoming telephone calls are handled in the following way:

- A terminal line may be attached to a modem instead of to a video or printing terminal. The modem plugs into a telephone outlet and permits the computer to answer incoming calls. This permits remote use of your computer from any location where there are a compatible terminal and modem. Such terminal lines have the status of shared terminals.

  In this mode of remote communication, the computer is open to any caller, authorized or unauthorized. The normal password security scheme is usually

1.7 An Internet Policy

From a security standpoint, an organization should declare which Internet services it is prepared to allow, and prohibit all other services. In order to do this, someone in the organization must be responsible for administering, and *understanding*, the services, which are selected.

The **Internet Policy** should have specifications, which cover the following issues:

- **The Services that are Connected**
  The services allowed should be explicitly stated, as should the direction (outward or inward) of communication permitted for each service.

  Users should be made aware of the preferred code of conduct for net users, often called *netiquette*, and should not abuse the facilities to which they have connection. They should be made aware that the Net connection is a business tool, and it should not be used to the detriment of the organization, which has provided it.

- **How the Services are Connected**
  The policy should mandate that all external services should exchange information with the internal network by means of a gateway, which should be configured and maintained only by an authorized administrator.
- **Who has Access to the Services**
  The users and the *machines*, which have authorized access to the Internet, should be known, and documented, and the security of the individual machines should be strong. In particular, those people with Internet access should maintain a secure password regime, and should set their machine to default to "password required" mode when they leave their machines unattended.
- **Intrusion Response Procedures**
  Those who are responsible for maintaining the Internet connection should be aware of the changing Internet vulnerabilities, what action they might be able to take to strengthen their gateway, how they would detect an intrusion or intrusion attempt, and what response they should make to a security event.

1.7.1 The Firewall

The usual solution to Internet security issues is to use a firewall as the device between and organization's internal network and the outside world. Although a firewall can be a single piece of hardware, it can also be a sub-system of routers, gateway-configured computers and server-configured computers. They are many ways to build security into your network's perimeter wall, with some fifty commercial ready-built products available, and a multitude of solutions for self-built systems, often referred to as the "belt-and-braces" approach, that I would be following till the end of my study.

sufficient to prevent unauthorized login, but a determined penetrator can sometimes breach it.

The following measures will reduce a penetrator's chances of breaching security by guesswork. Treat telephone numbers that your computer auto-answers as secret. If possible, change your system's tables (for example, *getty)* or the *login* utility so that the computer does not advertise its identity, or even which operating system it uses, until it authenticates the caller. Have a single telephone number to which users call. If possible, restrict use of this line to requests that the computer call back.

Outgoing telephone calls can also be made using the modem. In these cases, it is under the control of a process not logged in there, such as a daemon.

1.5 The OSI Model for Network Architecture

While TCP/IP is considered by many to be the *de facto* standard for network communications, the International Standards Organization (ISO) and the Comite Consultatif Internationale Telegraphique et Telephonique (CCITT) have been working since 1984 to define a reference model for Open Systems Interconnection (OSI) with security-related functionality. Protocols such as X.25 (packet-switching), X.400 (secure messaging) and X.500 (authentication and secure naming) have been produced, but as yet OSI has not displaced TCP/IP as the preferred network protocol suite for Wide Area Networks.

1.6 The Security Threat

Modems and network interfaces expose your computer system to hardware and to users not necessarily under your organization's control. For example, an email message from Asia to the Europe using the Internet may pass through a dozen different computer systems.

There are many different scenarios, which give rising levels of security threat, but there are the two basic considerations:

- Can you trust the "other" site - the site you are connected to?
- Are you satisfied with the integrity, availability and confidentiality of your network link?

If the answers to these questions indicate there are security risks, which must be addressed, then administrative action must be taken to install security measures which will reduce your risks to an acceptable level.

In particular, connection to the Internet should be done carefully, since this action takes connectivity to a new level- one in which there are potentially millions of "other" sites.

Chapter 2

The Internet and Internet Security

While Internet connectivity offers enormous benefits in terms of increased access to information, Internet connectivity is not necessarily a good thing for sites with low levels of security. The Internet suffers from glaring security problems that, if ignored, could have disastrous results for unprepared sites. An inherent problem with TCP/IP services, the complexity of host configuration, vulnerabilities introduced in the software development process, and a variety of other factors have all contributed to making unprepared sites open to intruder activity and related problems.

The following sections present a brief overview of the Internet, TCP/IP, and then explain what some of the Internet security related problems are and what factors have contributed to their seriousness.

2.1 Common Services

There are a number of services associated with TCP/IP and the Internet. The most commonly used service is electronic mail (e-mail), implemented by the Simple Mail Transfer Protocol (SMTP). Also, TELNET (terminal emulation), for remote terminal access, and FTP (file transfer protocol) are used widely. Beyond that, there are a number of services and protocols used for remote printing, remote file and disk sharing, management of distributed databases, and for information services. Following is a brief list of the most common services:

- **SMTP** - Simple Mail Transfer Protocol, used for sending and receiving electronic mail,
- **TELNET** - used for connecting to remote systems connected via the network, uses basic terminal emulation features.
- **FTP** - File Transfer Protocol, used to retrieve or store files on networked systems,
- **DNS** - Domain Name Service, used by TELNET, FTP, and other services for translating host names to IP addresses,
- Information-based services, such as
    - **Gopher** - a menu-oriented information browser and server that can provide a user-friendly interface to other information-based services,
    - **WAIS** - Wide Area Information Service, used for indexing and searching with databases of files, and
    - **WWW/http** - World Wide Web, a super-set of FTP, gopher, WAIS, other information services, using the hypertext transfer protocol (http), with Mosaic being a popular WWW client,
- **RPC-based services** - Remote Procedure Call services, such as

- **NFS** - Network File System, allows systems to share directories and disks, causes a remote directory or disk to appear to be local, and
- **NIS** - Network Information Services, allows multiple systems to share databases, e.g., the password file, to permit centralized management,

- **X Window System** - a graphical windowing system and set of application libraries for use on workstations, and
- **rlogin, rsh, and other "r" services** - employs a concept of mutually trusting hosts, for executing commands on other systems without requiring a password.

Although TCP/IP can be used equally well in a local area or wide area networking environment, a common use is for file and printer sharing at the local area networking level and for electronic mail and remote terminal access at both the local and the wide area networking levels

2.1.2 Internet Hosts

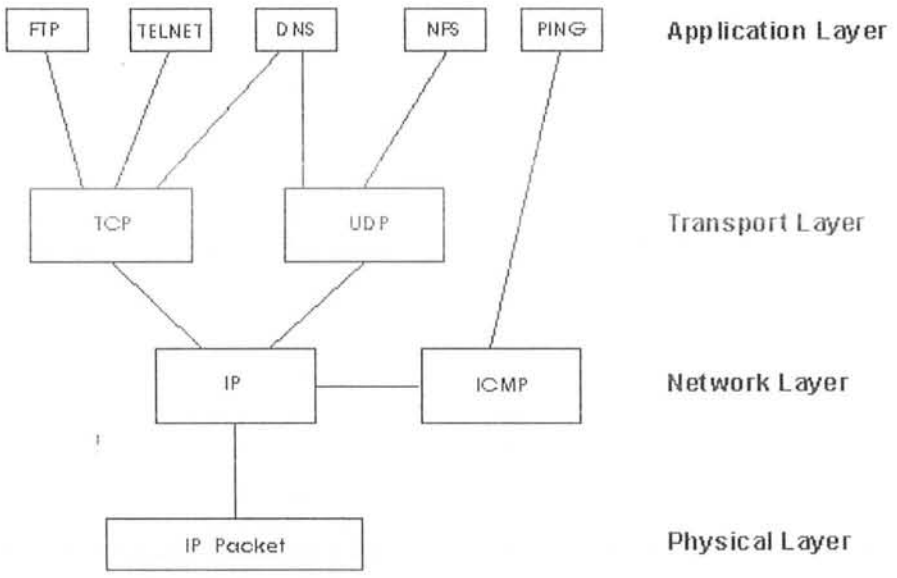
Many host systems connected to the Internet run a version of the UNIX operating system. TCP/IP was first implemented in the early 1980's for the version of UNIX written at the University of California at Berkeley known as the Berkeley Software Distribution (BSD). Many modern versions of UNIX derive there networking code directly from the BSD releases, thus UNIX provides a more-or-less standard set of TCP/IP services. This standard of sorts has resulted in many different versions of UNIX suffering from the same vulnerabilities, however it has also provided a common means for implementing firewall strategies such as IP packet filtering. It is important to note that BSD UNIX source code is fairly easy to obtain free from Internet sites, thus many good and bad people have been able to study the code for potential flaws and exploitable vulnerabilities.

Although UNIX is the predominant Internet host operating system, many other types of operating systems and computers are connected to the Internet, including systems running Digital Equipment Corporation's VMS, NeXT, mainframe operating systems, and personal computer operating systems such as for DOS, Microsoft Windows, and for Apple systems. Although personal computer systems often provide only client services, i.e., one can use TELNET to connect from but not to a personal computer, increasingly powerful personal computers are also beginning to provide, at low cost, the same services as larger hosts. Versions of UNIX for the personal computer, including Linux, FreeBSD, and BSD, and other operating systems such as Microsoft Windows NT, can provide the same services and applications that were, until recently, found only on larger systems. The ramifications of this are that more people are able to utilize a wider array of TCP/IP services than ever before. While this is good in that the benefits of networking are more available, it has negative consequences in that there is more potential for harm from intruders (as well as uneducated but well-intentioned users who, to some sites, may appear to be intruders).

2.2 Overview of TCP/IP Internals

Part of the popularity of the TCP/IP protocol suite is due to its ability to be implemented on top of a variety of communications channels and lower-level protocols such as T1 and X.25, Ethernet, and RS-232-controlled serial lines. Most sites use Ethernet connections at local area networks to connect hosts and client systems, and then connect that network via a T1 line to a regional network (i.e., a regional TCP/IP backbone) that connects to other organizational networks and backbones. Sites customarily have one connection to the Internet, but large sites often have two or more connections. Modem speeds are increasing as new communications standards are being approved, thus versions of TCP/IP that operate over the switched telephone network are becoming more popular. Many sites and individuals use PPP (Point-to-Point Protocol) and SLIP (Serial Line IP), to connect networks and workstations to other networks using the switched telephone network.

TCP/IP is more correctly a suite of protocols including TCP and IP, UDP (User Datagram Protocol), ICMP (Internet Control Message Protocol), and several others. The TCP/IP protocol suite does not conform exactly to the Open Systems Interconnection's seven layer model, but rather could be pictured as shown in this diagram:



2.2.1 IP

The IP layer receives packets delivered by lower-level layers, e.g., an Ethernet device driver, and passes the packets "up" to the higher-layer TCP or UDP layers. Conversely, IP transmits packets that have been received from the TCP or UDP layers to the lower-level layer.

IP packets are unreliable datagrams in that IP does nothing to ensure that IP packets are delivered in sequential order or are not damaged by errors. The IP packets contain the address of the host from which the packet was sent, referred to as the source address, and the address of the host that is to receive the packet, referred to as the destination address.

The higher-level TCP and UDP services generally assume that the source address in a packet is valid when accepting a packet. In other words, the IP address forms the basis of authentication for many services; the services trust that the packet has been sent from a valid host and that host is indeed who it says it is. IP does contain an option known as IP Source Routing, which can be used to specify a direct route to a destination and return path back to the origination. The route could involve the use of other routers or hosts that normally would not be used to forward packets to the destination. A source routed IP packet, to some TCP and UDP services, appears to come from the last system in the route as opposed to coming from the true origination. This option exists for testing purposes, however [Bel89] points out that source routing can be used to trick systems into permitting connections from systems that otherwise would not be permitted to connect. Thus, that a number of services trust and rely on the authenticity of the IP source address is problematic and can lead to breakins and intruder activity.

2.2.2 TCP

If the IP packets contain encapsulated TCP packets, the IP software will pass them "up" to the TCP software layer. TCP sequentially orders the packets and performs error correction, and implements virtual circuits, or connections between hosts. The TCP packets contain sequence numbers and acknowledgements of received packets so that packets received out of order can be reordered and damaged packets can be retransmitted.

TCP passes its information up to higher-layer applications, e.g., a TELNET client or server. The applications, in turn, pass information back to the TCP layer, which passes information down to the IP layer and device drivers and the physical medium, and back to the receiving host.

Connection oriented services, such as TELNET, FTP, rlogin, X Windows, and SMTP, require a high degree of reliability and therefore use TCP. DNS uses TCP in some cases (for transmitting and receiving domain name service databases), but uses UDP for transmitting information about individual hosts.

2.2.3 UDP

UDP interacts with application programs at the same relative layer as TCP. However, there is no error correction or retransmission of miss-ordered or lost packets. UDP is therefore not used for connection-oriented services that need a virtual circuit. It is used for services that are query-response oriented, such as NFS, where the number of messages with regard to the exchange is small compared to TELNET or FTP sessions. Services that use UDP include RPC-based services such as NIS and NFS, NTP (Network Time Protocol), and DNS (DNS also uses TCP).

It is easier to spoof UDP packets than TCP packets, since there is no initial connection setup (handshake) involved (since there is no virtual circuit between the two systems). Thus, there is a higher risk associated with UDP-based services.

2.2.4 ICMP

ICMP (Internet Control Message Protocol) is at the same relative layer as IP; its purpose is to transmit information needed to control IP traffic. It is used mainly to provide information about routes to destination addresses. ICMP redirect messages inform hosts about more accurate routes to other systems, whereas ICMP unreachable messages indicate problems with a route. Additionally, ICMP can cause TCP connections to terminate "gracefully" if the route becomes unavailable. Ping is a commonly used ICMP-based service. Older versions of UNIX could drop all connections between two hosts even if only one connection was experiencing network problems. Also, ICMP redirect messages can be used to trick routers and hosts acting as routers into using "false" routes; these false routes would aid in directing traffic to an attacker's system instead of a legitimate trusted system. This could in turn lead to an attacker gaining access to systems that normally would not permit connections to the attacker's system or network.

2.2.5 TCP and UDP Port Structure

TCP and UDP services generally have a client-server relationship. For example, a TEL NET server process initially sits idle at a system, waiting for an incoming connection. A user then interacts with a TELNET client process, which initiates a connection with the TELNET server. The client writes to the server, the server reads from the client and sends back its response. The client reads the response and reports back to the user. Thus, the connection is bi-directional and can be used for reading and writing.

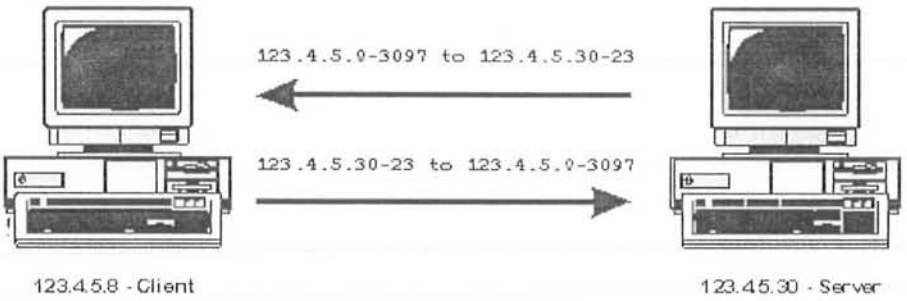
A TCP or UDP connection is uniquely identified by the following four items present in each message:

- **source IP address** - the address of the system that sent the packet,
- **destination IP address** - the address of the system that receives the packet,
- **source port** - the connection's port at the source system, and
- **destination port** - the connection's port at the destination system.

The port is a software construct that is used by the client or server for sending or receiving messages; a 1Bbit number identifies a port. Server processes are usually associated with a fixed port, e.g., 25 for SMTP or 6000 for X Windows; the port number is "well-known" because it, along with the destination IP address, needs to be used when initiating a connection to a particular host and service. Client processes, on the other hand, request a port number from the operating system when they begin execution; the port number is random although in some cases it is the next available port number.

As an example of how ports are used for sending and receiving messages, consider the TELNET protocol. The TELNET server listens for incoming messages on port 23, and

sends outgoing messages to port 23. A TELNET client, on the same or different system, would first request an unused port number from the operating system, and then use this port when sending and receiving messages. It would place this port number, say 3097, in packets destined for the TELNET server so that the server, when responding to the client, could place the client's port number in its TCP packets. The client's host, upon receiving a message, would examine the port and know which TELNET client should receive the message. This is shown conceptually in this diagram:



There is a somewhat uniform rule that only privileged server processes, i.e., those processes that operate with super-user privileges, can use port numbers less than 1024 (referred to as privileged ports). Servers mostly use ports numbered less than 1024, whereas clients generally must request unprivileged port numbers from the operating system. Although this rule is not firm and is not required in the TCP/IP protocol specifications, BSD-based systems adhere to it. As an accidental but fortuitous result, firewalls can block or filter access to services by examining the port numbers in TCP or UDP packets and then routing or dropping the packet based on a policy that specifies which services are permitted or denied.

Not all TCP and UDP servers and clients use ports in as straightforward a fashion as TELNET, but in general the procedure described here is useful in the firewalls context.

2.3 Security-Related Problems

As stated earlier, the Internet suffers from severe security-related problems. Sites that ignore these problems face some significant risk that intruders will attack them and that they may provide intruders with a staging ground for attacks on other networks. Even sites that do observe good security practices face problems with new vulnerabilities in networking software and the persistence of some intruders.

Some of the problems with Internet security are a result of inherent vulnerabilities in the services (and the protocols that the services implement), while others are a result of host configuration and access controls that are poorly implemented or overly complex to administer. Additionally, the role and importance of system management is often short changed in job descriptions, resulting in many administrators being, at best, part-time and poorly prepared. This is further aggravated by the tremendous growth of the Internet and how the Internet is used; businesses and agencies now depend on the Internet (often more

than they realize) for communications and research and thus have much more to lose if their sites are attacked. The following sections describe problems on the Internet and factors that contribute to these problems.

2.3.1 Weak Authentication

Incident handling teams estimate that many incidents stem from use of weak, static passwords. Passwords on the Internet can be "cracked" a number of different ways, however the two most common methods are by cracking the encrypted form of the password and by monitoring communications channels for password packets. The UNIX operating system usually stores an encrypted form of passwords in a file that can be read by normal users. The password file can be obtained by simply copying it or via a number of other intruder methods. Once the file is obtained, an intruder can run readily available password cracking programs against the passwords. If the passwords are weak, e.g., less that 8 characters, English words, etc., they could be cracked and used to gain access into the system. Another problem with authentication results from some TCP or UDP services being able to authenticate only to the granularity of host addresses and not to specific users. For example, an NFS (UDP) server cannot grant access to a specific user on a host, it must grant access to the entire host. The administrator of a server may trust a specific user on a host and wish to grant access to that user, but the administrator has no control over other users on that host and is thus forced to grant access to all users (or grant no access at all).

2.3.2 Ease of Spying/Monitoring

It is important to note that when a user connects to her account on a remote host using TELNET or FTP, the user's password travels across the Internet unencrypted, or in plaintext. Thus, another method for breaking into systems is to monitor connections for IP packets bearing a username and password, and then using them on the system to login normally. If the captured password is to an administrator account, then the job of obtaining privileged access is made much easier. As noted previously, hundreds and possibly thousands of systems across the Internet have been penetrated as a result of monitoring for usernames and passwords.

Electronic mail, as well as the contents of TELNET and FTP sessions, can be monitored and used to learn information about a site and its business transactions. Most users do not encrypt e-mail, yet many assume that e-mail is secure and thus safe for transmitting sensitive information.

The X Window System is an increasingly popular service that is also vulnerable to spying and monitoring. X permits multiple windows to be opened at a workstation, along with display of graphics and multi-media applications (for example, the WWW browser Mosaic). Intruders can sometimes open windows on other systems and read keystrokes that could contain passwords or sensitive information.

2.3.4 Ease of Spoofing

IP address of a host is presumed to be valid and is therefore trusted by TCP and UDP services. A problem is that, using IP source routing; an attacker's host can masquerade as a trusted host or client. Briefly, IP source routing is an option that can be used to specify a direct route to a destination and return path back to the origination. The route can involve the use of other routers or hosts that normally would not be used to forward packets to the destination. An example of how this can be used such that an attacker's system could masquerade as the trusted client of a particular server is as follows:

1. The attacker would change his host's IP address to match that of the trusted client,

2. The attacker would then construct a source route to the server that specifies the direct path the IP packets should take to the server and should take from the server back to the attacker's host, using the trusted client as the last hop in the route to the server,

3. The attacker sends a client request to the server using the source route,

4. The server accepts the client request as if it came directly from the trusted client and returns a reply to the trusted client,

5. The trusted client, using the source route, forwards the packet on to the attacker's host.

Many UNIX hosts accept source-routed packets and will pass them on as the source route indicates. Many routers will accept source routed packets as well, whereas some routers can be configured to block source routed packets.

An even simpler method for spoofing a client is to wait until the client system is turned off and then impersonate the client's system. In many organizations, staff members use personal computers and TCP/IP network software to connect to and utilize UNIX hosts as a local area network server. The personal computers often use NFS to obtain access to server directories and files (NFS uses IP addresses only to authenticate clients). An attacker could, after hours, configure a personal computer with the same name and IP address as another's, and then initiate connections to the UNIX host as if it were the "real" client. This is very simple to accomplish and likely would be an insider attack.

Electronic mail on the Internet is particularly easy to spoof and, without enhancements such as digital signatures, generally cannot be trusted. As a brief example, consider the exchange that takes place when Internet hosts exchange mail. The exchange takes place using a simple protocol consisting of ASCII-character commands. An intruder easily could enter these commands by hand by using TELNET to connect directly to a system's Simple Mail Transfer Protocol (SMTP) port. The receiving host trusts that the sending host is who it says it is, thus the origin of the mail can be spoofed easily by entering a sender address that is different from the true address. As a result, any user, without privileges, can falsify or spoof e-mail.

Other services, such as Domain Name Service, can be spoofed, but with more difficulty than electronic mail. These services still represent a threat that needs to be considered when using them.

2.3.5 Source Routing

Normally, the route a packet takes from its source to its destination is determined by the routers between the source and destination. The packet itself only says where it wants to go (the destination address), and nothing about how it expects to get there.

There is an optional way for the sender of a packet (the source) to include information in the packet that tells the route the packet should get to its destination; thus the name "source routing". For a firewall, source routing is noteworthy, since an attacker can generate traffic claiming to be from a system "inside" the firewall. In general, such traffic wouldn't route to the firewall properly, but with the source routing option, all the routers between the attacker's machine and the target will return traffic along the reverse path of the source route. Implementing such an attack is quite easy; so firewall builders should not discount it as unlikely to happen.

In practice, source routing is very little used. In fact, generally the main legitimate use is in debugging network problems or routing traffic over specific links for congestion control for specialized situations. When building a firewall, source routing should be blocked at some point. Most commercial routers incorporate the ability to block source routing specifically, and many versions of UNIX that might be used to build firewall bastion hosts have the ability to disable or ignore source-routed traffic.

2.3.6 ICMP Redirects & Redirect Bombs

An ICMP Redirect tells the recipient system to over-ride something in its routing table. It is legitimately used by routers to tell hosts that the host is using a non-optimal or defunct route to a particular destination, i.e. the host is sending it to the wrong router. The wrong router sends the host back an ICMP Redirect packet that tells the host what the correct route should be. If you can forge ICMP Redirect packets, and if your target host pays attention to them, you can alter the routing tables on the host and possibly subvert the security of the host by causing traffic to flow via a path the network manager didn't intend. ICMP Redirects also may be employed for denial of service attacks, where a host is sent a route that loses it connectivity, or is sent an ICMP Network Unreachable packet telling it that it can no longer access a particular network.
Many firewall builders screen ICMP traffic from their network, since it limits the ability of outsiders to ping hosts, or modify their routing tables.

2.3.7 Denial of Service

Denial of service is when someone decides to make your network or firewall useless by disrupting it, crashing it, jamming it, or flooding it. The problem with denial of service on the Internet is that it is impossible to prevent. The reason has to do with the distributed nature of the network: every network node is connected via other networks, which in turn

lonnect to other networks, etc. A firewall administrator or ISP only has control of a few of the local elements within reach. An attacker can always disrupt a connection "upstream" from where the victim controls it. In other words, if someone wanted to take a network off the air, they could do it either by taking the network off the air, or by taking the networks it connects to off the air, ad infinitum. There are many, many, ways someone can deny service, ranging from the complex to the brute-force.

2.3.8 Flawed LAN Services and Mutually Trusting Hosts

Host systems are difficult and time consuming to manage securely. To ease management demands and to enhance local area networking, some sites use services such as Network Information Services (NIS) and Network File System (NFS). These services can greatly reduce the amount of redundant management by permitting certain databases such as the password files to be managed in a distributed manner and by permitting systems to share files and data. Ironically, these services are inherently insecure and can be exploited to gain access by knowledgeable intruders. If a central server system is compromised, then the other systems trusting the central system could be compromised rather easily.

Some services such as rlogin allow for hosts to "trust" each other for the purposes of user convenience and enhanced sharing of systems and devices. If a system is penetrated or spoofed, and that system is trusted by other systems, it is simple for the intruder to then gain access to the other systems. As an example, a user with an account on more than one system can eliminate the need to enter a password at every system by configuring the accounts to trust connections from the user's primary system. When the user uses the rlogin command to connect to a host, the destination system will not ask for a password or account name, and the user's connection will be accepted. While this has a positive aspect in that the user's password does not get transmitted and could not be monitored and captured, it has a negative aspect in that if the user's primary account were to be penetrated, the intruder could simply use rlogin to penetrate the accounts on the other systems.

2.3.9 Complex Configuration and Controls

Host system access controls are often complex to configure and test for correctness. As a result, controls that are accidentally misconfigured can result in intruders gaining access. Some major UNIX vendors still ship host systems with access controls configured for maximum (i.e., least secure) access, which can result in unauthorized access if left as is.

A number of security incidents have occurred on the Internet due in part to vulnerabilities discovered by intruders (and subsequently, users, incident response teams, and vendors). Since most modern variants of UNIX derive their networking code from the BSD releases, and since the source code to the BSD releases is widely available, intruders have been able to study the code for bugs and conditions that can be exploited to gain access to systems. The bugs exist in part because of the complexity of the software and the inability to test it in all the environments in which it must operate. Sometimes the bugs

are easily discovered and corrected, other times little can be done except to rewrite the application, which is usually the option of last resort.

2.3.10 Host-based Security Does Not Scale

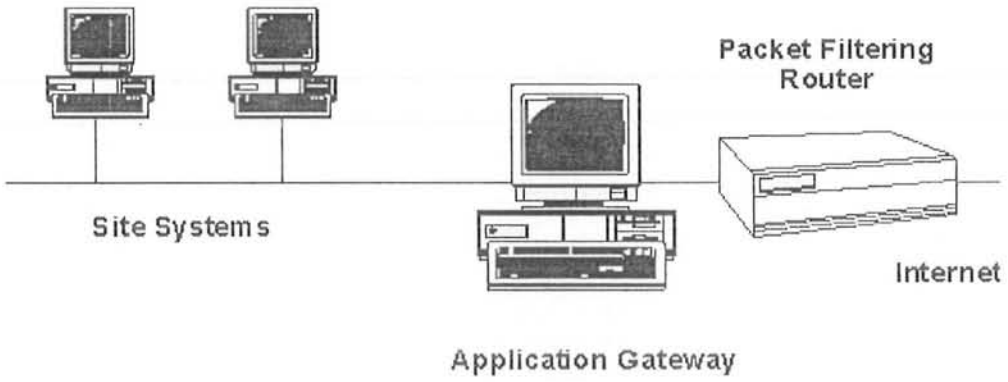
Host-based security does not scale well: as the number of hosts at a site increases, the ability to ensure that security is at a high level for each host decreases. Given that secure management of just one system can be demanding, managing many such systems could easily result in mistakes and omissions. A contributing factor is that the role of system management is often short-changed and performed in haste. As a result, some systems will be less secure than other systems, and these systems could be the weak links that ultimately will "break" the overall security chain.

If vulnerability is discovered in networking software, a site that is not protected by a firewall needs to correct the vulnerability on all exposed systems as quickly as possible. As discussed in Weak Authentication some vulnerability have permitted easy access to the UNIX root account; a site with many UNIX hosts would be particularly at risk to intruders in such a situation. Patching vulnerabilities on many systems in a short amount of time may not be practical and, if different versions of the operating system are in use, may not be possible.

Chapter 3

Introduction to Firewall

A number of the security problems with the Internet discussed earlier can be remedied or made less serious through the use of existing and well-known techniques and controls for host security. A firewall can significantly improve the level of site security while at the same time permitting access to vital Internet services.



3.1 The Firewall Concept

Perhaps it is best to describe first what a firewall is not: a firewall is not simply a router, host system, or collection of systems that provides security to a network. Rather, a firewall is an approach to security; it helps implement a larger security policy that defines the services and access to be permitted, and it is an implementation of that policy in terms of a network configuration, one or more host systems and routers, and other security measures such as advanced authentication in place of static passwords. The main purpose of a firewall system is to control access to or from a protected network (i.e., a site). It implements a network access policy by forcing connections to pass through the firewall, where they can be examined and evaluated.

A firewall system can be a router, a personal computer, a host, or a collection of hosts, set up specifically to shield a site or subnet from protocols and services that can be abused from hosts outside the subnet. A firewall system is usually located at a higher-level gateway, such as a site's connection to the Internet, however firewall systems can be located at lower-level gateways to provide protection for some smaller collection of hosts or subnets.

3.2 Why Firewalls?

The general reasoning behind firewall usage is that without a firewall, a subnet's systems expose themselves to inherently insecure services such as NFS or NIS and to probes and attacks from hosts elsewhere on the network. In a firewall-less environment, network security relies totally on host security and all hosts must, in a sense, co-operate to achieve a uniformly high level of security. The larger the subnet, the less manageable it is to maintain all hosts at the same level of security. As mistakes and lapses in security become more common, break-ins occur not as the result of complex attacks, but because of simple errors in configuration and inadequate passwords.

A firewall approach provides numerous advantages to sites by helping to increase overall host security. The following sections summarize the primary benefits of using a firewall.

3.2.1 Protection from Vulnerable Services

A firewall can greatly improve network security and reduce risks to hosts on the subnet by filtering inherently insecure services. As a result, the subnet network environment is exposed to fewer risks, since only selected protocols will be able to pass through the firewall.

For example, a firewall could prohibit certain vulnerable services such as NFS from entering or leaving a protected subnet. This provides the benefit of preventing the services from being exploited by outside attackers, but at the same time permits the use of these services with greatly reduced risk to exploitation. Services such as NIS or NFS that are particularly useful on a local area network basis can thus be enjoyed and used to reduce the host management burden.

Firewalls can also provide protection from routing-based attacks, such as source routing and attempts to redirect routing paths to compromised sites via ICMP redirects. A firewall could reject all source-routed packets and ICMP redirects and then inform administrators of the incidents.

3.2.2 Controlled Access to Site Systems

A firewall also provides the ability to control access to site systems. For example, some hosts can be made reachable from outside networks, whereas others can be effectively sealed off from unwanted access. A site could prevent outside access to its hosts except for special cases such as mail servers or information servers.

This brings to the fore an access policy that firewalls are particularly adept at enforcing: do not provide access to hosts or services that do not require access. Put differently, why provide access to hosts and services that could be exploited by attackers when the access is not used or required? If, for example, a user requires little or no network access to her desktop workstation, then a firewall can enforce this policy.

3.2.3 Concentrated Security

A firewall can actually be less expensive for an organization in that all or most modified software and additional security software could be located on the firewall systems as opposed to being distributed on many hosts. In particular, one-time password systems and other add-on authentication software could be located at the firewall as opposed to each system that needed to be accessed from the Internet.

Other solutions to network security such as Kerberos involve modifications at each host system. While Kerberos and other techniques should be considered for their advantages and may be more appropriate than firewalls in certain situations, firewalls tend to be simpler to implement in that only the firewall need run specialised software.

3.2.4 Enhanced Privacy

Privacy is of great concern to certain sites, since what would normally be considered innocuous information might actually contain clues that would be useful to an attacker. Using a firewall, some sites wish to block services such as finger and Domain Name Service. finger displays information about users such as their last login time, whether they've read mail, and other items. But, finger could leak information to attackers about how often a system is used, whether the system has active users connected, and whether the system could be attacked without drawing attention.

Firewalls can also be used to block DNS information about site systems, thus the names and IP addresses of site systems would not be available to Internet hosts. Some sites feel that by blocking this information, they are hiding information that would otherwise be useful to attackers.

3.2.5 Logging and Statistics on Network Use, Misuse

If all access to and from the Internet passes through a firewall, the firewall can log accesses and provide valuable statistics about network usage. A firewall, with appropriate alarms that sound when suspicious activity occurs can also provide details on whether the firewall and network are being probed or attacked.

It is important to collect network usage statistics and evidence of probing for a number of reasons. Of primary importance is knowing whether the firewall is withstanding probes and attacks, and determining whether the controls on the firewall are adequate. Network usage statistics are also important as input into network requirements studies and risk analysis activities.

3.2.6 Policy Enforcement

Lastly, but perhaps most importantly, a firewall provides the means for implementing and enforcing a network access policy. In effect, a firewall provides access control to users and services. Thus, a network access policy can be enforced by a firewall, whereas

without a firewall, such a policy depends entirely on the co-operation of users. A site may be able to depend on its own users for their co-operation, however it cannot nor should not depend on Internet users in general.

3.3 Issues and Problems with Firewalls

Given these benefits to the firewall approach, there are also a number of disadvantages, and there are a number of things that firewalls cannot protect against.

3.3.1 Restricted Access to Desirable Services

The most obvious disadvantage of a firewall is that it may likely block certain services that users want, such as TELNET, FTP, X Windows, NFS, etc. However, these disadvantage are not unique to firewalls; network access could be restricted at the host level as well, depending on a site's security policy. A well-planned security policy that balances security requirements with user needs can help greatly to alleviate problems with reduced access to services.

Some sites may have a topology that does not lend itself to a firewall, or may use services such as NFS in such a manner that using a firewall would require a major restructuring of network use. For example, a site might depend on using NFS and NIS across major gateways. In such a situation, the relative costs of adding a firewall would need to be compared against the cost of the vulnerabilities associated with not using a firewall, i.e., a risk analysis, and then a decision made on the outcome of the analysis. Other solutions such as Kerberos may be more appropriate, however these solutions carry their own disadvantages as well.

3.3.2 Large Potential for Back Doors

Secondly, firewalls do not protect against back doors into the site. For example, if unrestricted modem access is still permitted into a site protected by a firewall, attackers could effectively jump around the firewall. Modem speeds are now fast enough to make running SLIP (Serial Line IP) and PPP (Point-to-Point Protocol) practical; a SLIP or PPP connection inside a protected subnet is in essence another network connection and a potential backdoor.

3.3.3 Little Protection from Insider Attacks

Firewalls generally do not provide protection from insider threats. While a firewall may be designed to prevent outsiders from obtaining sensitive data, the firewall does not prevent an insider from copying the data onto a tape and taking it out of the facility. Thus, it is faulty to assume that the existence of a firewall provides protection from insider attacks or attacks in general that do not need to use the firewall. It is perhaps unwise to invest significant resources in a firewall if other avenues for stealing data or attacking systems are neglected.

3.3.4 Other Issues

Other problems or issues with firewalls are as follows:

- **WWW, gopher** - Newer information servers and clients such as those for World Wide Web (WWW), gopher, WAIS, and others were not designed to work well with firewall policies and, due to their newness, are generally considered risky. The potential exists for data-driven attacks, in which data processed by the clients can contain instructions to the clients; the instructions could tell the client to alter access controls and important security-related files on the host.
- **MBONE** - Multicast IP transmissions (MBONE) for video and voice are encapsulated in other packets; firewalls generally forward the packets without examining the packet contents. MBONE transmissions represent a potential threat if the packets were to contain commands to alter security controls and permit intruders.
- **Viruses** - Firewalls do not protect against users downloading virus-infected personal computer programs from Internet archives or transferring such programs in attachments to e-mail. Because these programs can be encoded or compressed in any number of ways, a firewall cannot scan such programs to search for virus signatures with any degree of accuracy. The virus problem still exists and must be handled with other policy and anti-viral controls.
- **Throughput** - Firewalls represent a potential bottleneck, since all connections must pass through the firewall and, in some cases, be examined by the firewall. However, this is generally not a problem today, as firewalls can pass data at T1 (1.5 Megabits/second) rates and most Internet sites are at connection rates less than or equal to T1.

3.4 Firewall Components

The primary components (or aspects) of a firewall are:

- **Network policy**,
- **Advanced authentication mechanisms**,
- **Packet filtering**, and
- **Application gateways**.

The following sections describe each of these components more fully.

3.4.1 Network Policy

There are two levels of network policy that directly influence the design, installation and use of a firewall system. The higher-level policy is an issue-specific, network access policy that defines those services that will be allowed or explicitly denied from the restricted network, how these services will be used, and the conditions for exceptions to this policy. The lower-level policy describes how the firewall will actually go about

restricting the access and filtering the services that were defined in the higher-level policy. The following sections describe these policies in brief.

3.4.1.1 Service Access Policy

The service access policy should focus on Internet-specific use issues as defined above, and perhaps all outside network access (i.e., dial-in policy, and SLIP and PPP connections) as well. This policy should be an extension of an overall organizational policy regarding the protection of information resources in the organization. For a firewall to be successful, the service access policy must be realistic and sound and should be drafted before implementing a firewall. A realistic policy is one that provides a balance between protecting the network from known risks, while still providing users access to network resources. If a firewall system denies or restricts services, it usually requires the strength of the service access policy to prevent the firewall's access controls from being modified on an ad hoc basis. Only a management-backed, sound policy can provide this.

A firewall can implement a number of service access policies, however a typical policy may be to allow no access to a site from the Internet, but allow access from the site to the Internet. Another typical policy would be to allow some access from the Internet, but perhaps only to selected systems such as information servers and e-mail servers. Firewalls often implement service access policies that allow some user access from the Internet to selected internal hosts, but this access would be granted only if necessary and only if it could be combined with advanced authentication.

3.4.1.2 Firewall Design Policy

The firewall design policy is specific to the firewall. It defines the rules used to implement the service access policy. One cannot design this policy in a vacuum isolated from understanding issues such as firewall capabilities and limitations, and threats and vulnerabilities associated with TCP/IP. Firewalls generally implement one of two basic design policies:

- Permit any service unless it is expressly denied, and
- Deny any service unless it is expressly permitted.

A firewall that implements the first policy allows all services to pass into the site by default, with the exception of those services that the service access policy has identified as disallowed. A firewall that implements the second policy denies all services by default, but then passes those services that have been identified as allowed. This second policy follows the classic access model used in the areas of information security.

The first policy is less desirable, since it offers more avenues for getting around the firewall, e.g., users could access new services currently not denied by the policy (or even addressed by the policy) or run denied services at non-standard TCP/UDP ports that aren't denied by the policy. Certain services such as X Windows, FTP, Archie, and RPC cannot be filtered easily, and are better accommodated by a firewall that implements the first policy. The second policy is stronger and safer, but it is more difficult to implement

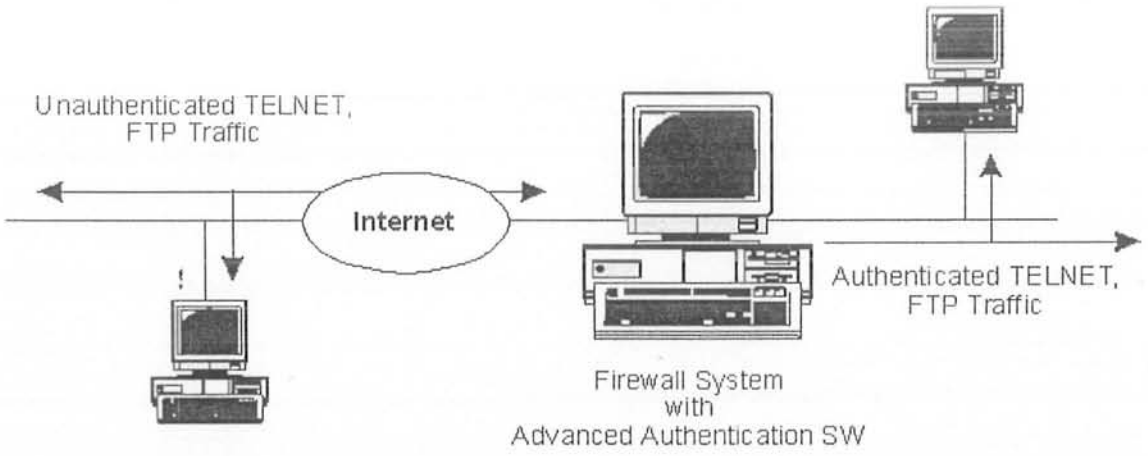
and may impact users more in that certain services such as those just mentioned may have to be blocked or restricted more heavily.

The service access policy is the most significant component of the four described here. The other three components are used to implement and enforce the policy. The effectiveness of the firewall system in protecting the network depends on the type of firewall implementation used, the use of proper firewall procedures, and the service access policy.

3.4.2 Advanced Authentication

Advanced authentication measures such as smartcards, authentication tokens, biometrics, and software-based mechanisms are designed to counter the weaknesses of traditional passwords. While the authentication techniques vary, they are similar in that the passwords generated by advanced authentication devices cannot be reused by an attacker who has monitored a connection.

Some of the more popular advanced authentication devices in use today are called one-time password systems. A smartcard or authentication token, for example, generates a response that the host system can use in place of a traditional password. Because the token or card works in conjunction with software or hardware on the host, the generated response is unique for every login. The result is a one-time password that, if monitored, cannot be reused by an intruder to gain access to an account.



Since firewalls can centralize and control site access, the firewall is the logical place for the advanced authentication software or hardware to be located.

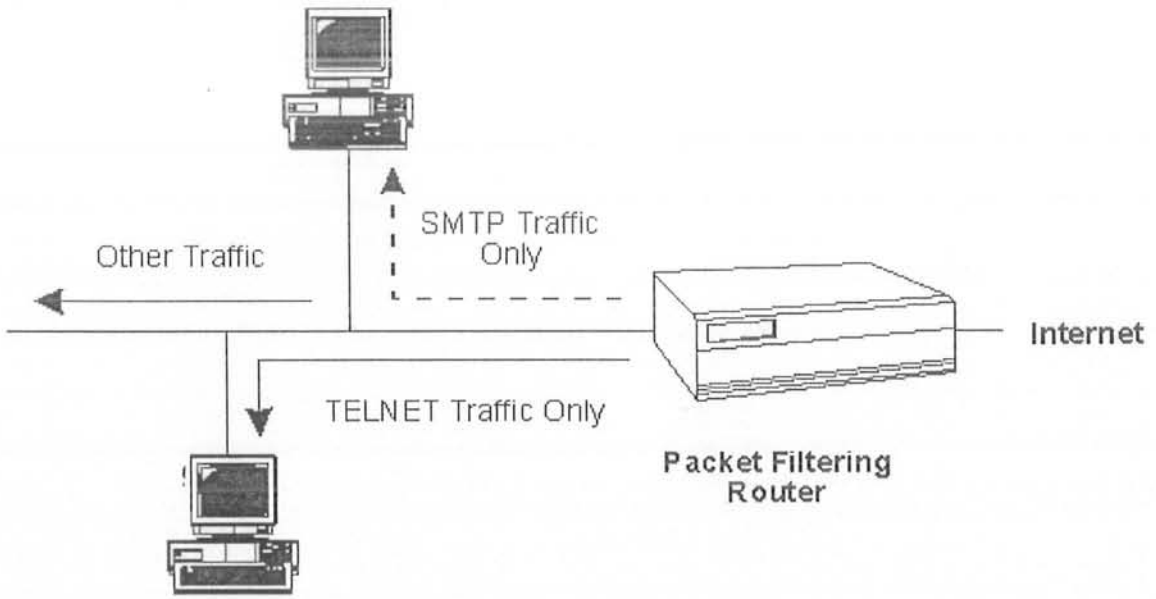
3.4.3 Packet Filtering

IP packet filtering is done usually using a packet filtering router designed for filtering packets as they pass between the router's interfaces. A packet filtering router usually can filter IP packets based on some or all of the following fields:

- Source IP address,
- Destination IP address,
- TCP/UDP source port, and
- TCP/UDP destination port.

Filtering can be used in a variety of ways to block connections from or to specific hosts or networks, and to block connections to specific ports. A site might wish to block connections from certain addresses, such as from hosts or sites that it considers to be hostile or untrustworthy. Alternatively, a site may wish to block connections from all addresses external to the site (with certain exceptions, such as with SMTP for receiving e-mail) .

Adding TCP or UDP port filtering to IP address filtering results in a great deal of flexibility. Servers such as the TELNET daemon reside usually at specific ports, such as port 23 for TELNET. If a firewall can block TCP or UDP connections to or from specific ports, then one can implement policies that call for certain types of connections to be made to specific hosts, but not other hosts. For example, a site may wish to block all incoming connections to all hosts except for several firewalls-related systems. At those systems, the site may wish to allow only specific services, such as SMTP for one system and TELNET or FTP connections to another system. With filtering on TCP or UDP ports, this policy can be implemented in a straightforward fashion by a packet filtering router or by a host with packet filtering capability.



3.4.3.1 Which Protocols to Filter

The decision to filter certain protocols and fields depends on the network access policy, i.e., which systems should have Internet access and the type of access to permit. The

following services are inherently vulnerable to abuse and are usually blocked at a firewall from entering or leaving the site:

- **tftp**, port 69, trivial FTP, used for booting diskless workstations, terminal servers and routers, can also be used to read any file on the system if set up incorrectly,
- **X Windows, Open Windows**, ports 6000+, port 2000, can leak information from X window displays including all keystrokes,
- **RPC**, port 111, Remote Procedure Call services including NIS and NFS, which can be used to steal system information such as passwords and read and write to files, and
- **rlogin, rsh, and rexec**, ports 513, 514, and 512, services that if improperly configured can permit unauthorized access to accounts and commands.

Other services, whether inherently dangerous or not, are usually filtered and possibly restricted to only those systems that need them. These would include:

- **TELNET**, port 23, often restricted to only certain systems,
- **FTP**, ports 20 and 21, like TELNET, often restricted to only certain systems,
- **SMTP**, port 25, often restricted to a central e-mail server,
- **RIP**, port 520, routing information protocol, can be spoofed to redirect packet routing,
- **DNS**, port 53, domain names service zone transfers, contains names of hosts and information about hosts that could be helpful to attackers, could be spoofed,
- **gopher, http** , ports 70 and 80, information servers and client programs for gopher and WWW clients, should be restricted to an application gateway that contains proxy services.

While some of these services such as TELNET or FTP are inherently risky, blocking access to these services completely may be too drastic a policy for many sites. Not all systems, though, generally require access to all services. For example, restricting TELNET or FTP access from the Internet to only those systems that require the access can improve security at no cost to user convenience.

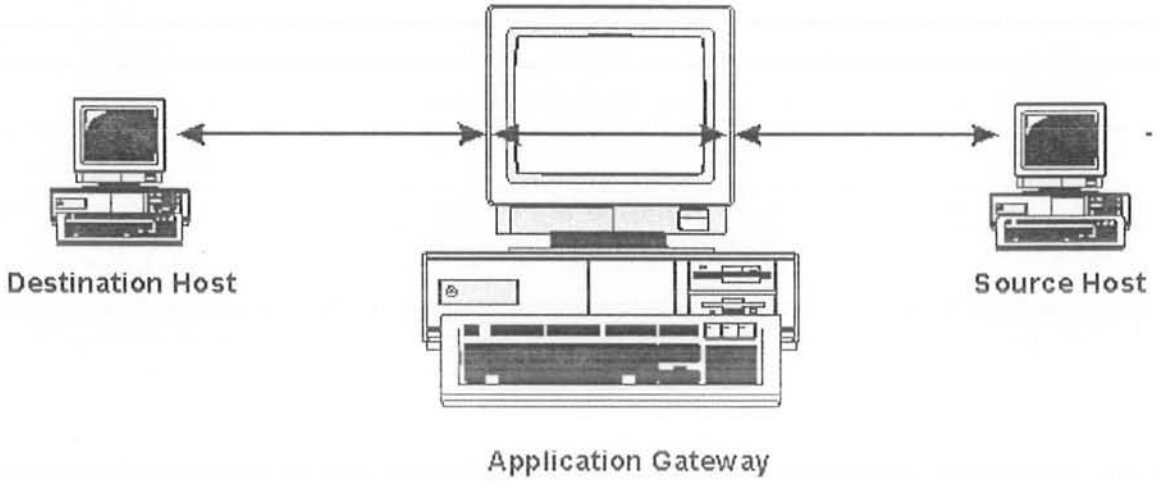
3.4.4 Application Gateways

To counter some of the weaknesses associated with packet filtering routers, firewalls need to use software applications to forward and filter connections for services such as TELNET and FTP. Such an application is referred to as a proxy service, while the host running the proxy service is referred to as an application gateway. Application gateways and packet filtering routers can be combined to provide higher levels of security and flexibility than if either were used alone.

As an example, consider a site that block all incoming TELNET and FTP connections using a packet filtering router. The router allows TELNET and FTP packets to go to one host only, the TELNET/FTP application gateway. A user who wishes to connect inbound

to a site system would have to connect first to the application gateway, and then to the destination host, as follows:

1. A user first telnets to the application gateway and enters the name of an internal host,
2. The gateway checks the user's source IP address and accepts or rejects it according to any access criteria in place,
3. The user may need to authenticate herself (possibly using a one-time password device),
4. The proxy service creates a TELNET connection between the gateway and the internal host,
5. The proxy service then passes bytes between the two connections, and
6. The application gateway logs the connection.



Application gateways have a number of general advantages over the default mode of permitting application traffic directly to internal hosts. These include:

- **Information hiding**, in which the names of internal systems need not necessarily be made known via DNS to outside systems, since the application gateway may be the only host whose name must be made known to outside systems,
- **Robust authentication and logging**, in which the application traffic can be pre-authenticated before it reaches internal hosts and can be logged more effectively than if logged with standard host logging,
- **Cost-effectiveness**, because third-party software or hardware for authentication or logging need be located only at the application gateway, and
- **Less-complex filtering rules**, in which the rules at the packet filtering router will be less complex than they would if the router needed to filter application traffic and direct it to a number of specific systems. The router need only allow application traffic destined for the application gateway and reject the rest.

3.4.5 Circuit-Level Gateways

A circuit-level gateway relays TCP connections but does no extra processing or filtering of the protocol. For example, the TELNET application gateway example provided here would be an example of a circuit-level gateway, since once the connection between the source and destination is established, the firewall simply passes bytes between the systems

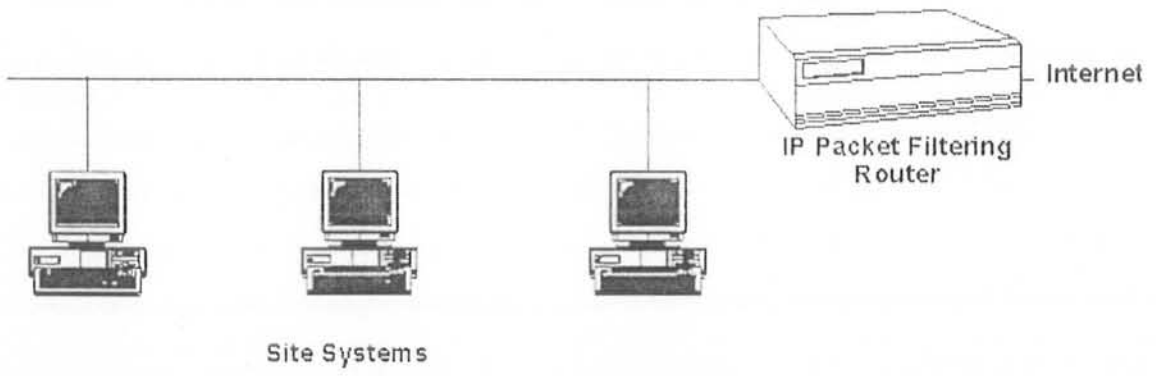
3.5 Firewall Architecture

There are lots of ways to structure the network to protect systems using a firewall. Some of them are as follows:

- Packet Filtering Firewall,
- Dual-homed Gateway Firewall,
- Screened Host Firewall, and
- Screened Subnet Firewall.

3.5.1 Packet Filtering Firewall

Basically, one installs a packet filtering router at the Internet (or any subnet) gateway and then configures the packet filtering rules in the router to block or filter protocols and addresses. The site systems usually have direct access to the Internet while all or most access to site systems from the Internet is blocked. However, the router could allow selective access to systems and services, depending on the policy. Usually, inherently dangerous services such as NIS, NFS, and X Windows are blocked.

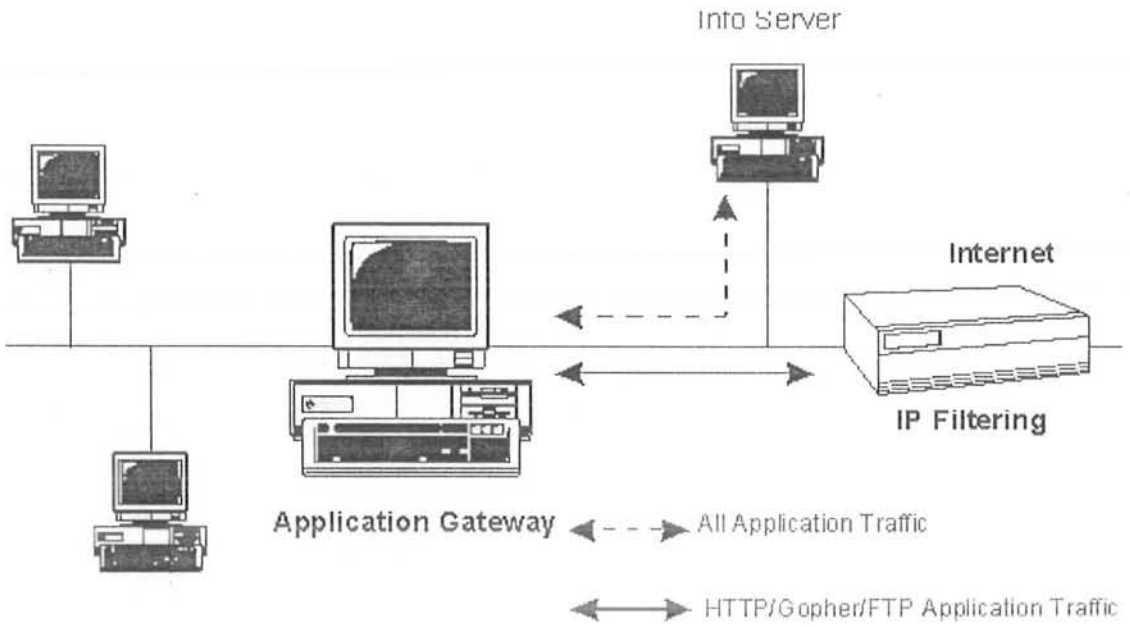


3.5.2 Dual-homed Gateway Firewall

The dual-homed gateway, shown below, is a better alternative to packet filtering router firewalls. It consists of a host system with two network interfaces, and with the host's IP forwarding capability disabled (i.e., the default condition is that the host can no longer route packets between the two connected networks). In addition, a packet filtering router can be placed at the Internet connection to provide additional protection. This would

create an inner, screened subnet that could be used for locating specialized systems such as information servers and modem pools.

Unlike the packet filtering firewall, the dual-homed gateway is a complete block to IP traffic between the Internet and protected site. Services and access is provided by proxy servers on the gateway. It is a simple firewall, yet very secure.



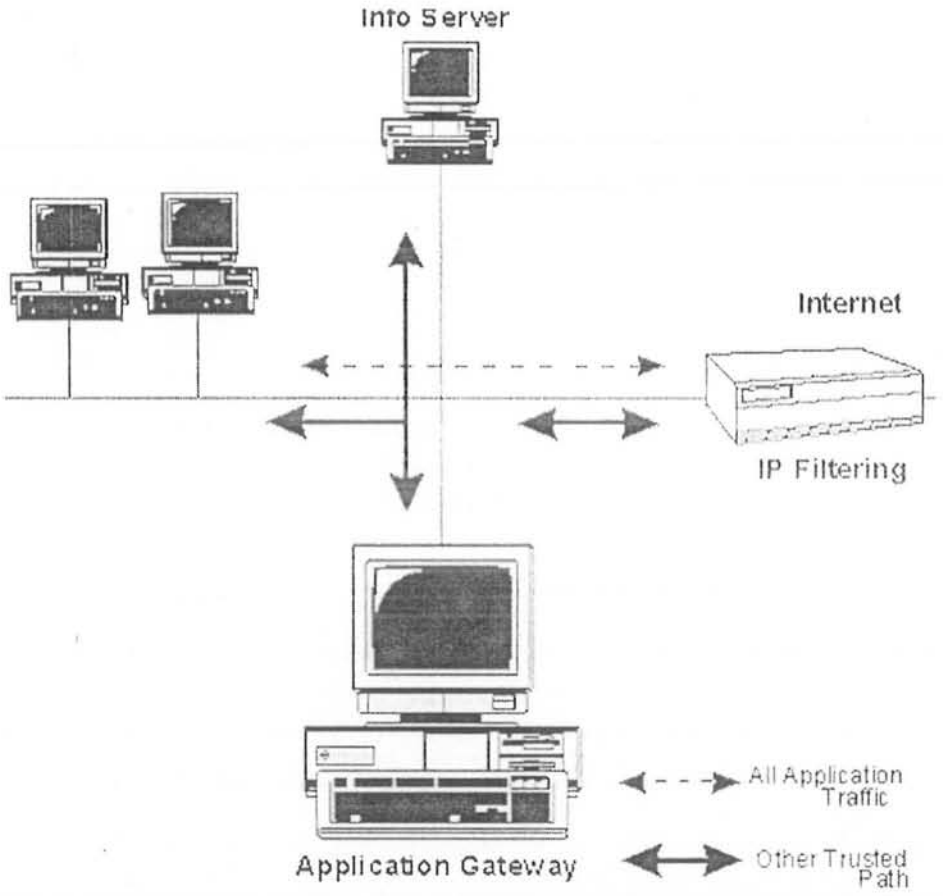
This type of firewall implements the second design policy, i.e., deny all services unless they are specifically permitted, since no services pass except those for which proxies exist. The ability of the host to accept source-routed packets would be disabled, so that no other packets could be passed by the host to the protected subnet. It can be used to achieve a high degree of privacy since routes to the protected subnet need to be known only to the firewall and not to Internet systems (because Internet systems cannot route packets directly to the protected systems). The names and IP addresses of site systems would be hidden from Internet systems, because the firewall would not pass DNS information.

3.5.3 Screened Host Firewall

The screened host firewall, shown below, is a more flexible firewall than the dual-homed gateway firewall, however the flexibility is achieved with some cost to security. The screened host firewall is often appropriate for sites that need more flexibility than that provided by the dual-homed gateway firewall. The screened host firewall combines a packet-filtering router with an application gateway located on the protected subnet side of the-router.

The application gateway needs only one network interface. The application gateway's proxy services would pass TELNET, FTP, and other services for which proxies exist, to site systems. The router filters or screens inherently dangerous protocols from reaching the application gateway and site systems. It rejects (or accepts) application traffic according to the following rules:

- Application traffic from Internet sites to the application gateway gets routed,
- All other traffic from Internet sites gets rejected, and
- The router rejects any application traffic originating from the inside unless it came from the application gateway.



Unlike the dual-homed gateway firewall, the application gateway needs only one network interface and does not require a separate subnet between the application gateway and the router. This permits the firewall to be made more flexible but perhaps less secure by permitting the router to pass certain trusted services "around" the application gateway and directly to site systems. The trusted services might be those for which proxy services don't exist, and might be trusted in the sense that the risk of using the services has been considered and found acceptable.

3.5.4 Screened Subnet Firewall

The screened subnet firewall is a variation of the dual-homed gateway and screened host firewalls. It can be used to locate each component of the firewall on a separate system, thereby achieving greater throughput and flexibility, although at some cost to simplicity. But, each component system of the firewall needs to implement only a specific task, making the systems less complex to configure.

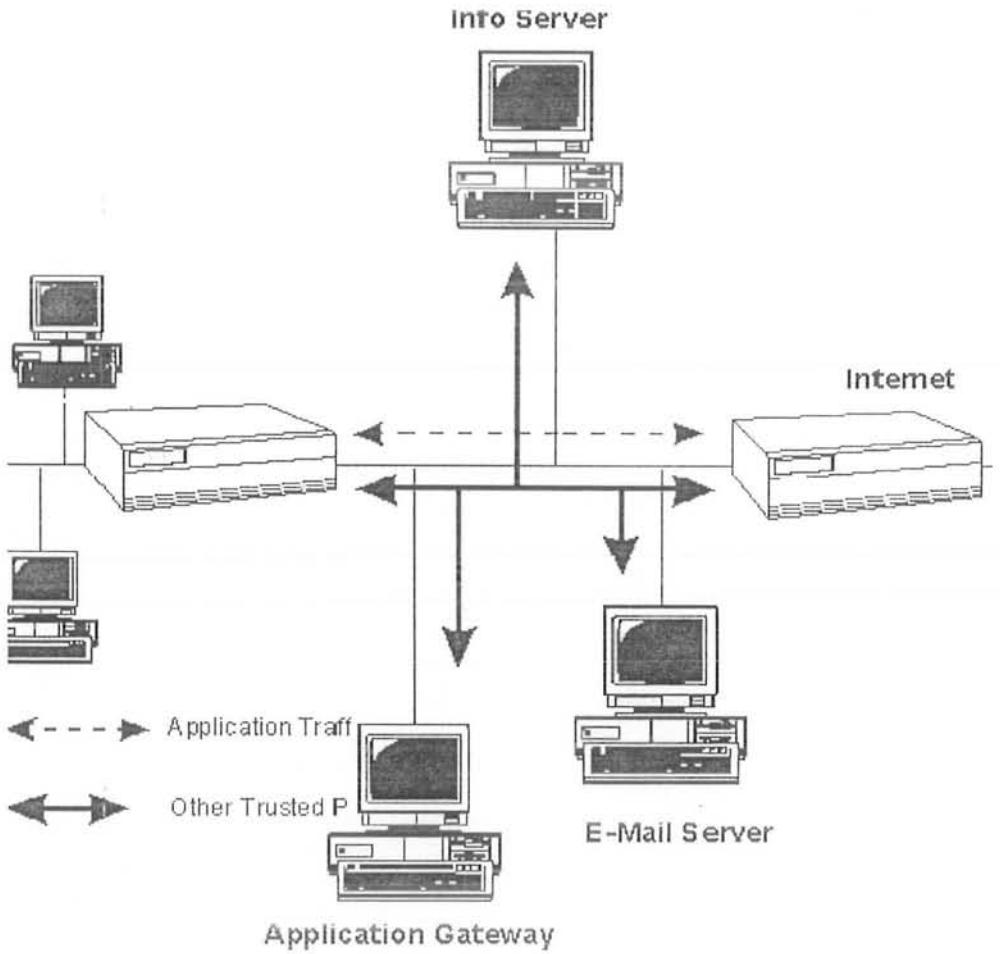
In the *Screened Subnet Firewall* two routers are used to create an inner, screened subnet. This subnet sometimes referred as the "DMZ", houses the application gateway, however it could also house information servers, modem pools, and other systems that require carefully-controlled access. The router shown as the connection point to the Internet would route traffic according to the following rules:

- application traffic from the application gateway to Internet systems gets routed,
- e-mail traffic from the e-mail server to Internet sites gets routed,
- application traffic from Internet sites to the application gateway gets routed,
- e-mail traffic from Internet sites to the e-mail server gets routed,
- ftp, http, etc., traffic from Internet sites to the information server gets routed, and
- all other traffic gets rejected.

The outer router restricts Internet access to specific systems on the screened subnet, and blocks all other traffic to the Internet originating from systems that should not be originating connections (such as the modem pool, the information server, and site systems). The router would be used as well to block packets such as NFS, NIS, or any other vulnerable protocols that do not need to pass to or from hosts on the screened subnet.

The inner router passes traffic to and from systems on the screened subnet according to the following rules:

- Application traffic from the application gateway to site systems gets routed,
- E-mail traffic from the e-mail server to site systems gets routed,
- Application traffic to the application gateway from site systems get routed,
- E-mail traffic from site systems to the e-mail server gets routed,
- ftp, http, etc., traffic from site systems to the information server gets routed,
- All other traffic gets rejected.

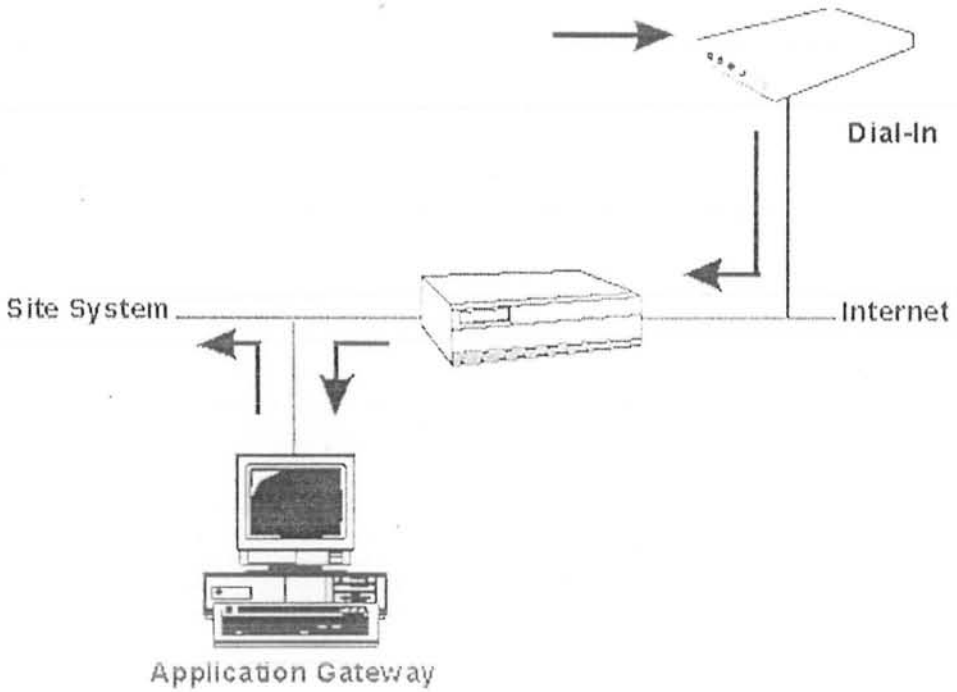


Thus, no site system is directly reachable from the Internet and vice versa, as with the dual-homed gateway firewall. A big difference, though, is that the routers are used to direct traffic to specific systems, thereby eliminating the need for the application gateway to be dual-homed. Greater throughput can be achieved, then, if a router is used as the gateway to the protected subnet. Consequently, the screened subnet firewall may be more appropriate for sites with large amounts of traffic or sites that need very high-speed traffic. The two routers provide redundancy in that an attacker would have to subvert both routers to reach site systems directly. The application gateway, e-mail server, and information server could be set up such that they would be the only systems "known" from the Internet; no other system name need be known or used in a DNS database that would be accessible to outside systems

3.6 Integrating Modem Pools with Firewalls

Many sites permit dial-in access to modems located at various points throughout the site. This is a potential backdoor and could negate all the protection provided by the firewall. A much better method for handling modems is to concentrate them into a modem pool, and then secure connections from that pool.

The modem pool likely would consist of modems connected to a terminal server, which is a specialized computer designed for connecting modems to a network. A dial-in user connects to the terminal server, and then connects (e.g., telnets) from there to other host systems. Some terminal servers provide security features that can restrict connections to specific systems, or require users to authenticate using an authentication token. Alternatively, the terminal server can be a host system with modems connected to it.



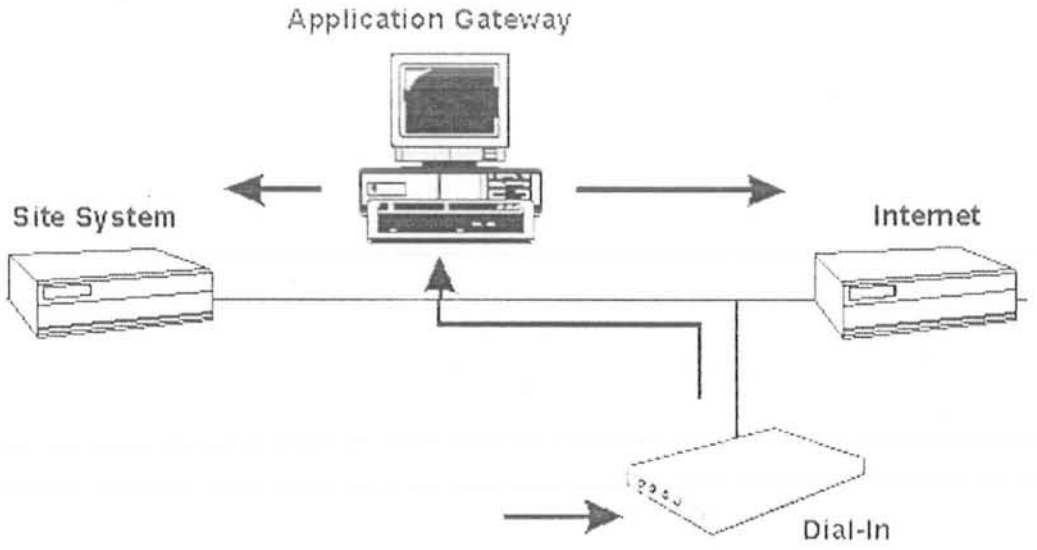
Above figure shows a modem pool located on the Internet side of the screened host firewall. Since the connections from modems need to be treated with the same suspicion as connections from the Internet, locating the modem pool on the outside of the firewall forces the modem connections to pass through the firewall.

The application gateway's advanced authentication measures can be used then to authenticate users who connect from modems as well as from the Internet. The packet filtering router could be used to prevent inside systems from connecting directly to the modem pool.

A disadvantage to this, though, is that the modem pool is connected directly to the Internet and thus more exposed to attack. If an intruder managed to penetrate the modem pool, the intruder might use it as a basis for connecting to and attacking other Internet systems. Thus, a terminal server with security features to reject dial-in connections to any system but the application gateway should be used.

The dual-homed gateway and screened subnet firewalls provide a more secure method for handling modem pools. In following figure the terminal server gets located on the inner, screened subnet, where access to and from the modem pool can be carefully controlled by

the routers and application gateways. The router on the Internet side protects the modem pool from any direct Internet access except from the application gateway.



With the dual-homed gateway and screened subnet firewalls, the router connected to the Internet would prevent routing between Internet systems and the modem pool. With the screened subnet firewall, the router connected to the site would prevent routing between site systems and the modem pool; with the dual-homed gateway firewall, the application gateway would prevent the routing. Users dialing into the modem pool could connect to site systems or the Internet only by connecting to the application gateway, which would use advanced authentication measures.

Chapter 4

The Netfilter framework in Linux 2.4

4.1 Netfilter basics / concepts

Over the past several years, the use of Linux as a firewall platform has grown significantly. Linux firewalling code has come a long way since the time ipfwadm was introduced in kernel 1.2. Recent changes in linux firewalling code include netfilter architecture (controlled from the command line by iptables utility), which was introduced in stable kernel 2.4. The newest version 2.4 of Linux kernel presents many new security enhancements such as: enhanced capabilities, better support for encryption (for VPN and encrypted file systems) and netfilter architecture, which is a re-implementation of Linux's firewalling code and which remains fully backward-compatible due to the use of ipchains and ipfwadm loadable kernel modules.

4.2 What is netfilter?

Netfilter is definitely more than any of the firewall subsystems in the past linux kernels. Netfilter provides a abstract, generalized framework of which one particular incarnation is the packet filtering subsystem.

The netfilter framework consists out of three parts:

1. Each protocol defines a set of 'hooks' (IPv4 defines 5), which are well-defined points in a packet's traversal of that protocol stack. At each of these points, the protocol stack will call the netfilter framework with the packet and the hook number.
2. Parts of the kernel can register to listen to the different hooks for each protocol. So when a packet is passed to the netfilter framework, it checks to see if anyone has registered for that protocol and hook; if so, they get a chance to examine (and possibly alter) the packet, discard it, allow it to pass or ask netfilter to queue the packet for userspace.
3. Packets that have been queued are collected for sending to userspace; these packets are handled asynchronously. A userspace process can examine the packet, can alter it, and reinject it at the same hook it left the kernel.

All the packet filtering / NAT / ... stuff is based on this framework. There is no more dirty packet altering code spread all over the network stack.

The netfilter framework currently has been implemented for IPv4, IPv6 and DECnet.

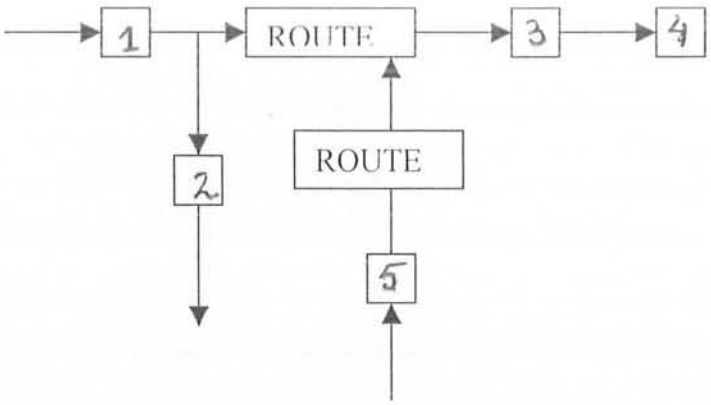
4.3 Why did we need netfilter?

Following points explains why Iptables was needed instead of Ipchains.

- No infrastructure for passing packets to userspace, so all code which does some packet fiddling must be done as kernel code. Kernel programming is hard, must be done in C, and is dangerous.
- Transparent proxying is extremely difficult We have to look up every packet to see if there's a socket bound to that adderess. No clean interface.
- Creating of packet filter rules independent of interface address is impossible. We must know local interface address to distinguish locally-generated or locally-terminated packets from through packets. The forward chain has only information on outgoing interface. So we must try to figure out where the packet came from.
- Masquerading and packet filtering are implemented as one part This makes the firewalling code way too complex.
- Ipchains code is neither modular nor extensible (eg. for MAC adress filtering)

4.4 Netfilter architecture in IPv4

A Packet Traversing the Netfilter System:



Packets come in from the left. After verification of the IP checksum, the packets hit the NF_IP_PRE_ROUTING [1] hook.

Next they enter the routing code, which decides if the packets are local or have to be passed to another interface.

If the packets are considered to be local, they traverse th NF_IP_LOCAL_IN [2] hook and get passed to the process (if any) afterwards.

If the packets are routed to another interface, they pass the NF_IP_FORWARD [3] hook.

The packet passes a final netfilter hook. NF_IP_POST_ROUTING [4], before they get transmitted on the target interface.

The NF_IP_LOCAL_OUT [5] hook is called for locally generated packets. Here You can see that routing occurs after this hook is called: in fact, the routing code is called first (to figure out the source IP address and some IP options), and called again if the packet is altered.

Locally generated packets hit NF_IP_POST_ROUTING [4], too.

4.4.1 Netfilter base

Kernel modules can register for one or more of this hook and get called for each packet traversing the hook. The module is free to alter the packet and returns netfilter one of these values:

- NF_ACCEPT continue traversal as normal
- NF_DROP drop the packet; do not continue traversal
- NF_STOLEN I've taken over the packet; do not continue traversal
- NF_QUEUE queue the packet (usually for userspace handling)
- NF_REPEAT call this hook again

4.4.2 Packet selection: IP tables

A packet selection system called IP tables has been built based on the netfilter framework. It is a direct descendant of ipchains, with extensibility.

Kernel modules can register a new table, and ask for a packet to traverse a given table. This packet selection is used for packet filtering (the 'filter' table), Network Address Translation (the 'nat' table) and general packet mangling (the 'mangle' table).

The three big parts of Linux 2.4 packet handling are built using netfilter hooks and IP tables. They are seperate modules and are independent from each other. They all plug in nicely in the infrastructure provided by netfilter.

Packet filtering This table 'filter' should never alter packets, only filter them. One of the advantages of iptables over ipchains is that it is small and fast, and it hooks into netfilter at the NF_IP_LOCAL_IN, NF_IP_FORWARD and NF_IP_LOCAL_OUT hooks.

Therefore, for each packet there is one, and only one, place to filter it. This is one big change compared to ipchains, where a forwarded packet used to traverse three chains.

NAT The nat table listens at three netfilter hooks: NF_IP_PRE_ROUTING and NF_IP_POST_ROUTING to do source and destination NAT for routed packets. For destination altering of local packets, the NF_IP_LOCAL_OUT hook is used.

This table is different from the 'filter' table, in that only the first packet of a new connection will traverse the table. The result of this traversal is then applied to all future packets of the same connection.

The NAT table is used for source NAT, destination NAT, masquerading (which is a special case of source nat) and transparent proxying (which is a special case of destination nat).

Packet mangling The 'mangle' table registers at the NF_IP_PRE_ROUTING and NF_IP_LOCAL_OUT hooks.

Using the mangle table You can modify the packet itself or some of the out-of-band data attached to the packet. Currently the alteration of the TOS bits as well as setting the nfmark field inside the skb is implemented on top of the mangle table.

Connection tracking is fundamental to NAT, but has been implemented as a seperate module. This allows an extension to the packet filtering code to simply use connection tracking for "stateful firewalling". (the 'state' match) .

Chapter 5

Packet filtering using iptables

5.1 Traversing of tables and chains

When a packet first enters the firewall, it hits the hardware and then gets passed on to the proper device driver in the kernel. Then the packet starts to go through a series of steps in the kernel, before it is either sent to the correct application (locally), or forwarded to another host - or whatever happens to it. In this example, we're assuming that the packet is destined for another host on another network. The packet goes through the different steps in the following fashion:

Forwarded packets

Step	Table	Chain	Comment
1			On the wire (i.e., internet)
2			Comes in on the interface (i.e., eth0)
3	mangle	PREROUTING	This chain is normally used for mangling packets, i.e., changing TOS and so on.
4	nat	PREROUTING	This chain is used for Destination Network Address Translation mainly. Source Network Address Translation is done further on. Avoid filtering in this chain since it will be bypassed in certain cases.
5			Routing decision, i.e., is the packet destined for our localhost or to be forwarded and where.
6	filter	FORWARD	The packet gets routed onto the FORWARD chain. Only forwarded packets go through here, and here we do all the filtering. Note that all traffic that's forwarded goes through here (not only in one direction), so you need to think about it when writing your ruleset.
7	nat	POSTROUTING	This chain should first and foremost be used for Source Network Address Translation. Avoid doing filtering here, since certain packets might pass this chain without ever hitting it. This is also where Masquerading is done.

Step	Table	Chain	Comment
8			Goes out on the outgoing interface (i.e., eth1).
9			Out on the wire again (i.e., LAN).

As seen above, there are quite a lot of steps to pass through. The packet can be stopped at any of the **iptables** chains, or anywhere else if it is malformed Do note that there are no specific chains or tables for different interfaces or anything like that. FORWARD is always passed by all packets that are forwarded over this firewall/router. INPUT is meant solely for packets to our local host that does not get routed to any other destination.

Now, let us have a look at a packet that is destined for our own localhost. It would pass through the following steps before actually being delivered to our application that receives it:

Destination local host (our own machine)

Step	Table	Chain	Comment
1			On the wire (e.g., Internet)
2			Comes in on the interface (e.g., eth0)
3	mangle	PREROUTING	This chain is normally used for mangling packets, i.e., changing TOS and so on.
4	Nat	PREROUTING	This chain is used for Destination Network Address Translation mainly. Avoid filtering in this chain since it will be bypassed in certain cases.
5			Routing decision, i.e., is the packet destined for our local host or to be forwarded and where.
6	Filter	INPUT	This is where we do filtering for all incoming traffic destined for our localhost. Note that all incoming packets destined for this host pass through this chain, no matter what interface or in which direction they came from.
7			Local process/application (i.e., server/client program)

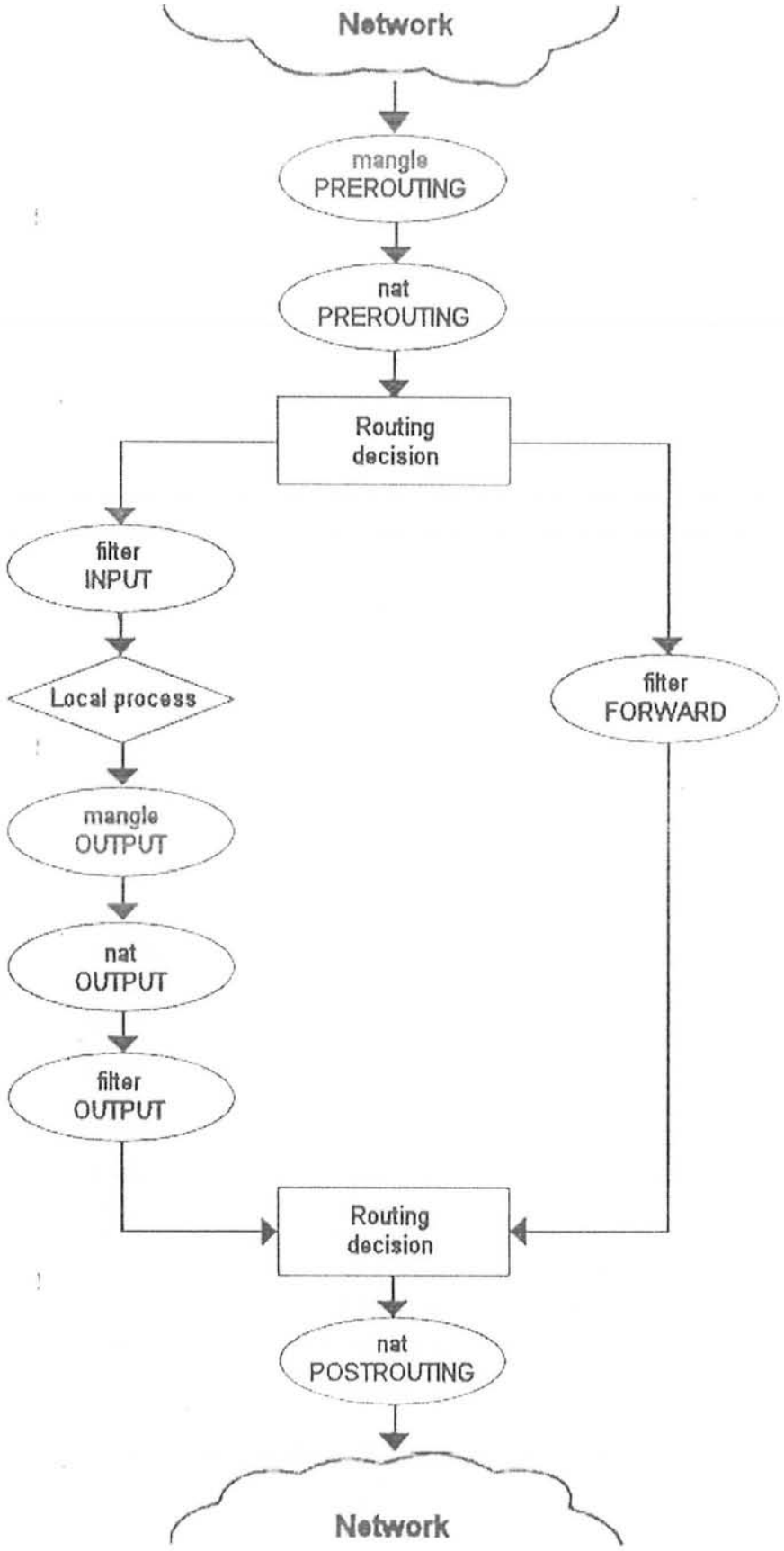
Note that this time the packet was passed through the INPUT chain instead of the FORWARD chain.

Finally we look at the outgoing packets from our own local host and what steps they go through.

Source local host (our own machine)

Step	Table	Chain	Comment
1			Local process/application (i.e., server/client (program)
2	Mangle	OUTPUT	This is where we mangle packets, it is suggested that you do not filter in this chain since it can have side effects.
3	Nat	OUTPUT	No
4	Filter	OUTPUT	This is where we filter packets going out from the local host.
5			Routing decision. This is where we decide where the packet should go.
6	Nat	POSTROUTING	This is where we do Source Network Address Translation as described earlier. It is suggested that you don't do filtering here since it can have side effects, and certain packets might slip through even though you set a default policy of **DROP**.
7			Goes out on some interface (e.g., eth0)
8			On the wire (e.g., Internet)

Following figure tries to show what's been happening:


Network
mangle
PREROUTING
nat
PREROUTING
Routing
decision
filter
INPUT
Local process
mangle
OUTPUT
nat
OUTPUT
filter
OUTPUT
filter
FORWARD
Routing
decision
nat
POSTROUTING
Network

5.2 Mangle table

This table is mainly used for mangling packets. Targets that are only valid in the mangle table:

- TOS
- TTL
- MARK

The **TOS** target is used to set and/or change the Type of Service field in the packet. This could be used for setting up policies on the network regarding how a packet should be routed and so on. Note that this has not been perfected and is not really implemented on the internet and most of the routers don't care about the value in this field, and sometimes, they act faulty on what they get.

The **TTL** target is used to change the TTL (Time To Live) field of the packet. We could tell packets to only have a specific TTL and so on.

The **MARK** target is used to set special mark values to the packet. These marks could then be recognized by the **iproute2** programs to do different routing on the packet depending on what mark they have, or if they don't have any. We could also do bandwidth limiting and Class Based Queuing based on these marks.

5.3 Nat table

This table should only be used for NAT (Network Address Translation) on different packets. In other words, it should only be used to translate the packet's source field or destination field. Only the first packet in a stream will hit this chain. After this, the rest of the packets will automatically have the same action taken on them as the first packet. The actual targets that do these kinds of things are:

- DNAT
- SNAT
- MASQUERADE

The **DNAT** (Destination Network Address Translation) target is mainly used in cases where you a public IP and want to redirect accesses to the firewall to some other host (on a DMZ for example). In other words, we change the destination address of the packet and reroute it to the host.

SNAT (Source Network Address Translation) is mainly used for changing the source address of packets. For the most part you'll hide your local networks or DMZ, etc. A very good example would be that of a firewall of which we know outside IP address, but need to substitute our local network's IP numbers whit that of our firewall. With this target the firewall will automatically **SNAT** and **De-SNAT** the packets, hence making it possible to make connections from the LAN to the Internet. If our network uses 192.168.0.0/netmask

for example, the packets would never get back from the Internet, because IANA has regulated these networks (amongst others) as private and only for use in isolated LANs.

The **MASQUERADE** target is used in exactly the same way as **SNAT**, but the **MASQUERADE** target takes a little bit more overhead to compute. The reason for this is that each time that the **MASQUERADE** target gets hit by a packet, it automatically checks for the IP address to use, instead of doing as the **SNAT** target does - just using the single configured IP address. The **MASQUERADE** target makes it possible to work properly with Dynamic DHCP IP addresses that your ISP might provide for your PPP, or SLIP connections to the internet.

5.4 Filter table

The filter table is, of course, mainly used for filtering packets. We can match packets and filter them in whatever way we want. There is nothing special to this chain or to packets that might slip through because they are malformed, etc. This is the place that we actually take action against packets and look at what they contain and **DROP** or /**ACCEPT** them, depending on their payload. Of course we may also do prior filtering; however, this particular table, is the place for which filtering was designed. Almost all targets are usable in this chain; however, the targets discussed previously can only be used in their respective tables.

Chapter 6

The state machine

6.1 Introduction

This section will deal with the state machine and explain it in detail.

The state machine is a special part within iptables that should really not be called the state machine at all, since it is really a connection tracking machine. Connections tracking is done to let the Netfilter framework know the state of a specific connection. Firewalls that implement this are generally called stateful firewalls. A stateful firewall is generally much more secure than non-stateful firewalls since it allows us to write much tighter rulesets.

Within iptables, the state of a connection can be divaded in 4 different basic states. These are known as **NEW**, **ESTABLISHED**, **RELATED** and **INVALID**. With the **--state** match we can specify which states that we want to allow in or out. The conntrack module keeps all states freshly in memory, according to certain rules on when to release a certain state and on specific information. The connection tracking module calculates the state of a specific packet upon 4 basic values in TCP and UDP and then on a few extra values. The basic values used to calculate a state for TCP and UDP streams are: The source IP address, the destination IP address, the source port and, lastly, the destination port. For ICMP packages, other rules apply. The same goes for other subprotocols of the IP protocol.

Previous Linux kernels had the possibility to turn on and off defragmentation. However, since iptables and Netfilter were introduced and connection tracking in particular, this option was gotten rid of. The reason for this, is that connection tracking can not work properly without defragmenting packets, and hence defragmenting has been incorporated to conntrack and is carried out automatically. It can not be turned off, except by turning off connection tracking. Defragmentation is always carried out if connection tracking is turned on.

All connection tracking is handled in the PREROUTING chain. What this means, is that iptables will do all recalculation of states and so on, within the PREROUTING chain. If we send the initial packet in a stream, this is where the state gets set to **NEW**, and when we receive a return packet, this is where the state is changed to **ESTABLISHED**, and so on.

6.2 The conntrack entries

Let's take a brief look at a conntrack entry and how to read them in /proc/net/ip_conntrack. This gives a list of all the current entries in conntrack database. If the ip_conntrack module is loaded, a **cat** of /proc/net/ip_conntrack might look like:

tcp 6 117 SYN_SENT src=192.168.1.6 dst=192.168.1.9 sport=32775 dport=22 [UNREPLIED] src=192.168.1.9 dst=192.168.1.6 sport=22 dport=32775 use=2

This example contains all the information that the conntrack module maintains to know which state a specific connection is in. First of all, we have a protocol, which in this case is tcp. Next, the same value in normal decimal coding. After this, we see how long this conntrack entry has to live. This value is set to 117 seconds right now and is decremented regularly until we see more traffic. This value is then reset to the default value for the specific state that it is in at that relevant point of time. Next comes the actual state that this entry is in at the present point of time. In the above mentioned case we are looking at a packet that is in the SYN_SENT state. The internal value of a connection is slightly different from the ones used externally with **iptables**. The value SYN_SENT tells us that we are looking at a connection that has only seen a TCP SYN packet in one direction. Next, we see the source IP address, destination IP address, source port and destination port. At this point we see a specific keyword that tells us that we have seen no return traffic for this connection. Lastly, we see what we expect of return packets. The information details the source IP address and destination IP address (which are both inverted, since the packet is to be directed back to us). The same thing goes for the source port and destination port of the connection. These are the values that should be of any interest to us.

The connection tracking entries may take on a series of different values, all specified in the conntrack headers available in linux/include/netfilter-ipv4/ip_conntrack*.h files. These values are dependant on which subprotocol of IP we use. TCP, UDP or ICMP protocols take specific default values as specified in linux/include/netfilter-ipv4/ip_conntrack.h. Also, depending on how this state changes, the default value of the time until the connection is destroyed will also change.

When a connection has seen traffic in both directions, the conntrack entry will erase the [UNREPLIED] flag, and then reset it. The entry tells us that the connection has not seen any traffic in both directions, will be replaced by the [ASSURED] flag, to be found close to the end of the entry. The [ASSURED] flag tells us that this connection is assured and that it will not be erased if we reach the maximum possible tracked connections. Thus, connections marked as [ASSURED] will not be erased, contrary to the non assured connections (those not marked as [ASSURED]). How many connections that the connection tracking table can hold depends upon a variable that can be set through the ipsysctl functions in recent kernels. The default value held by this entry varies heavily depending on how much memory is available. On 128 MB of RAM , 8192 entries and at

256 MB of RAM, 16376 entries are possible. Settings can be read and set through the /proc/sys/net/ipv4/ip_conntrack_max setting.

6.3 Userland states

As seen, packets may take on several different states within the kernel itself, depending on what protocol we are talking about. However, outside the kernel, we only have the 4 states as described previously. These states can mainly be used in conjunction with the state match which will then be able to match packets based on their current connection tracking state. The valid states are **NEW**, **ESTABLISHED**, **RELATED** and **INVALID** states. The following table will briefly explain each possible state.

Userland states

State	Explanation
NEW	The **NEW** state tells us that the packet is new in the connection. This means that the first packet that the conntrack module sees will be matched.
ESTABLISHED	The **ESTABLISHED** state has seen traffic in both directions and will then match those packets. **ESTABLISHED** connections are fairly easy to understand. The only requirement to get into an **ESTABLISHED** state is that one host sends a packet, and that it later on gets a reply from the other host. The **NEW** state will upon receipt to the firewall change to the **ESTABLISHED** state.
RELATED	The **RELATED** state is one of the more tricky states. A connection is considered **RELATED** when it is related to another already **ESTABLISHED** connection. What this means, is that for a connection to be considered as **RELATED**, we must first have a connection that is considered **ESTABLISHED**. The **ESTABLISHED** connection will then spawn a connection outside of the main connection. The newly spawned connection will then be considered **RELATED**, if the conntrack module is able to understand that it is **RELATED**. ICMP error messages and redirects etc can be considered as **RELATED** if we have generated a packet that in turn generated the ICMP message.
INVALID	The **INVALID** state means that the packet can not be identified or that it does not have any state. This may be due to several reasons, such as the system running out of memory or ICMP error messages that do not respond to any known connections. Generally, it is a good idea to **DROP** everything in this state.

These states can be used together with the **--state** match to match packets based on their connection tracking state. This is what makes the state machine so incredibly strong and efficient for our firewall. Previously, we often had to open up all ports above 1024 to let all traffic back into our local networks again. With the state machine in place this is not

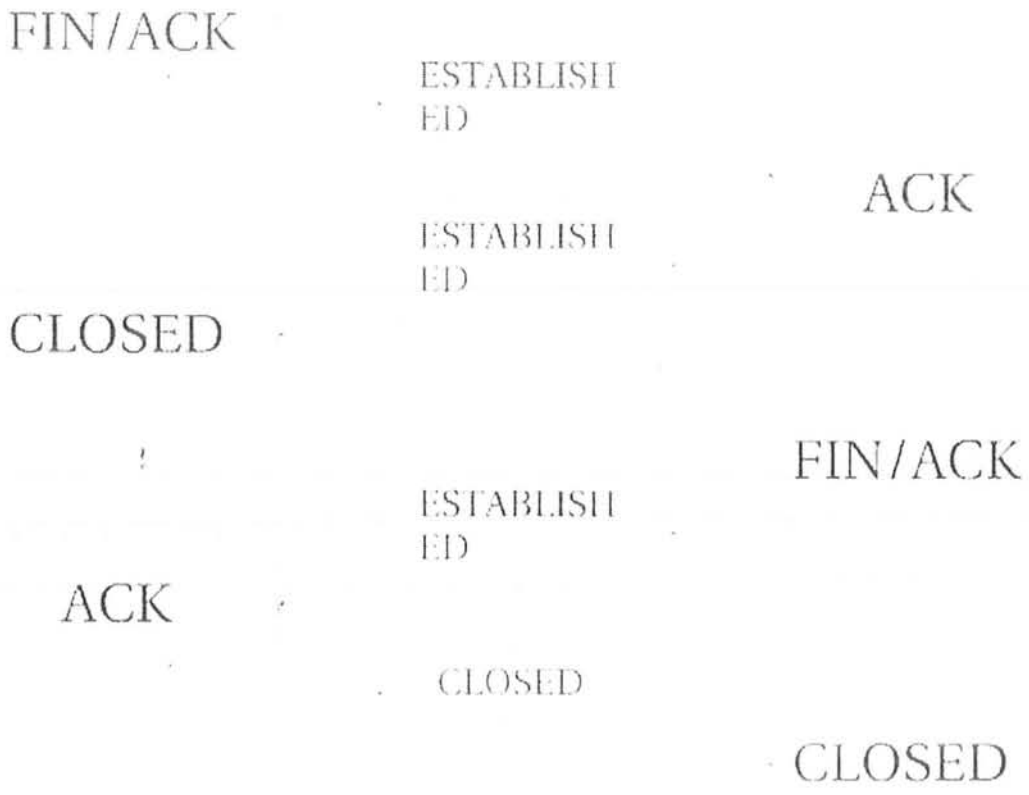
necessary any longer, since we can now just open up the firewall for return traffic and not for all kinds of other traffic.

6.4 TCP connections

A TCP connection is always initiated with the 3-way handshake, which establishes and negotiates the actual connection over which data will be sent. The whole session is begun with a SYN packet, then a SYN/ACK packet and finally an ACK packet to acknowledge the whole session establishment. At this point the connection is established and able to start sending data.

As far as the user is concerned, connection tracking works basically the same for all connection types. Picture above shows exactly what state the stream enters during the different stages of the connection. As seen, the connection tracking code does not really follow the flow of the TCP connection, from the users' viewpoint. Once it has seen one packet (the SYN), it considers the connection as NEW. Once it sees the return packet (SYN/ACK), it considers the connection as **ESTABLISHED**. With this particular implementation, **NEW** and **ESTABLISHED** packets can be allowed to leave local network, only **ESTABLISHED** connections are allowed back, and that will work perfectly. Conversely, if the connection tracking machine were to consider the whole connection establishment as **NEW**, we would never really be able to stop outside connections to our local network, since we would have to allow **NEW** packets back in again.

When a TCP connection is closed down, it is done in the following way and takes the following states.



As seen above, the connection is never really closed until the last ACK is sent. This picture only describes how it is closed down under normal circumstances. A connection may also, for example, be closed by sending a RST(reset), if the connection were to be refused. In this case, the connection would be closed down after a predetermined time.

When the TCP connection has been closed down, the connection enters the TIME_WAIT state, which is per default set to 2 minutes. This is used so that all packets that have gotten out of order can still get through our ruleset, even after the connection has already closed. This is used as a kind of buffer time so that packets that have gotten stuck in one or another congested router can still get to the firewall, or to the other end of the connection.

If the connection is reset by a RST packet, the state is changed to CLOSE. This means that the connection per default have 10 seconds before the whole connection is definitely closed down. RST packets are not acknowledged in any sense, and will break the connection directly. There are also other states than the ones told so far. Here is the complete list of possible states that a TCP stream may take, and their timeout values.

Internal states

State	Timeout value
NONE	30 minutes
ESTABLISHED	5 days
SYN_SENT	2 minutes
SYN_RECV	60 seconds
FIN_WAIT	2 minutes
TIME_WAIT	2 minutes
CLOSE	10 seconds
CLOSE_WAIT	12 hours
LAST_ACK	30 seconds
LISTEN>	2 minutes

These values are most definitely not absolute. They may change with kernel revisions, and they may also be changed via the proc filesystem in the /proc/sys/net/ipv4/netfilter/ip_ct_tcp_* variables. These values are set in 1/100th parts of seconds, so 3000 means 30 seconds.

6.5 UDP connections

UDP connections are in themself not stateful connections, but rather stateless. This is so for several reasons, mainly it does not contain any connection establishment or connection closing, but most of all it lacks sequencing. Receiving two UDP datagrams in a specific order does not say anything about which order they where sent in. It is however still possible to set states on the connections within the kernel.

6.6 ICMP connections

ICMP packets are far from a stateful stream since they are only used for controlling and should never establish any connections. There are 4 ICMP types that will generate return packets however, and these have 2 different states. These ICMP messages can take the **NEW** and **ESTABLISHED** states. These ICMP types are Echo request and reply, Timestamp request and reply, Information request and reply and finally Address mask request and reply. Out of these, the timestamp request and information request are obsolete and could most probably just be dropped. However, the Echo messages are used in several setups such as pinging hosts.

As seen see in the above picture, the server sends a echo request to the client, which is considered as **NEW** by the firewall. The client then responds with a echo reply which the firewall considers as state **ESTABLISHED**.

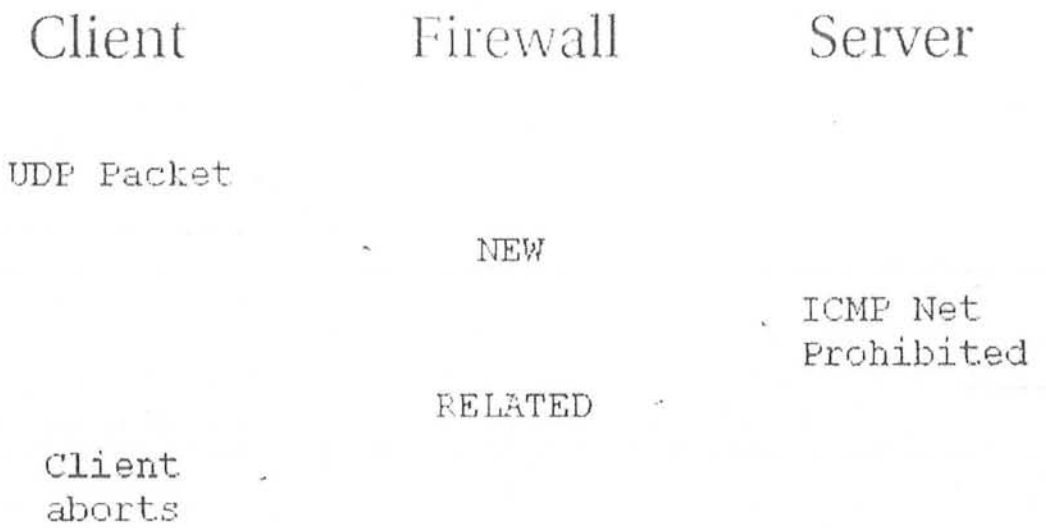
The reply packet is considered as **ESTABLISHED**. However, after the ICMP reply, there will be absolutely no more legal traffic in the same connection. For this reason, the connection tracking entry is destroyed once the reply has traveled all the way through the netfilter structure.

ICMP requests has a default timeout of 30 seconds, which can be changed in the /proc/sys/net/ipv4/netfilter/ip_ct_icmp_timeout entry.

Another huge part of ICMP is the fact that they are used to tell the hosts what happened to specific UDP and TCP connections or connection attempts. For this simple reason, they will very often be recognized as **RELATED** to original connections. A simple example would be the ICMP Host unreachable or ICMP Network unreachable. In this case, the ICMP reply is considered as a **RELATED** packet. The following picture should explain how it would look.

In the above example, we send out a SYN packet to a specific address. This is considered as a **NEW** connection by the firewall. However, the network the packet is trying to reach is unreachable, so a router returns a network unreachable ICMP error to us. The connection tracking code can recognize this packet as **RELATED**.So the ICMP reply is correctly sent to the client which will then abort. Meanwhile, the firewall has destroyed the connection tracking entry since it knows this was an error message.

The same behaviour as above is experienced with UDP connections if they run into any problem like the above. All ICMP messages sent in reply to UDP connections are considered as **RELATED**. Consider the following picture:



This time an UDP packet is sent to the server. This UDP connection is considered as **NEW**. However, the network is administratively prohibited by some firewall or router on the way over. Hence, our firewall receives a ICMP Network Prohibited in return. The

firewall knows that this ICMP error message is related to the already opened UDP connection and sends it as an **RELATED** packet to the client. At this point, the firewall destroys the connection tracking entry, and the client receives the ICMP message and should abort.

6.7 Default connections

In certain cases, the conntrack entries does not know how to handle a specific protocol. This happens as soon as it does not know about the protocol in specific or how it works. In these cases, it goes back to a default behaviour. The default behaviour is used on, for example, NETBLT, MUX and EGP. The default behaviour looks pretty much the same as the UDP connection tracking. The first packet is considered **NEW**, and reply traffic and so forth is considered **ESTABLISHED**.

When the default behaviour is used, all of these packets will use the same default timeout value. This can be set via the /proc/sys/net/ipv4/netfilter/ip_ct_generic_timeout variable. The default value here is 600 seconds, or 6 minutes.

Chapter 7

How a rule is built

This chapter will discuss in length how to build rules. A rule could be described as the pure rules the firewall will follow when blocking different connections and packets in each chain. Each line that's inserted to a chain should be considered a rule.

7.1 Basics

Each rule is a line that the kernel looks at to find out what to do with a packet. If all the criteria, or matches, are met, we perform the target, or jump, instruction. Normally we would write a rule something like this:

iptables [-t *table*] command [match] [target/jump]

The **-t** option specifies which table to use. Per default, the filter table is used. If another table than the standard table is to be used, you could insert the table specification where [table] is specified. However, it is not necessary to specify it explicitly all the time since **iptables** per default uses the filter table to implement your commands on. It is not required to put the table specification at this location, either. It could be set pretty much anywhere in the rule, however, it is more or less standard to put the table specification at the beginning of the command line.

The command should always be first or directly after the table specification. This tells the **iptables** command what to do. This first variable is used to tell the program what to do, for example to insert a rule or to add a rule to the end of the chain, or to delete a rule.

The match is the part which we send to the kernel that says what a packet must look like to be matched. We could specify what IP address the packet must come from, or which network interface the packet must come from etc.

Finally we have the target of the packet. If all the matches are met for a packet we tell the kernel to perform this action on the packet. We could tell the kernel to send the packet to another chain that we create our self, which must be part of this table. We could tell the kernel to drop this packet and do no further processing, or we could tell kernel to send a specified reply to be sent back.

7.2 Commands

The command tells **iptables** what to do with the rest of the command line that we send to the program. Normally we want to either add or delete something to some table or another. The following commands are available to iptables:

Commands

Command	**-A, --append**
Example	**iptables -A INPUT ...**
Explanation	This command appends the rule to the end of the chain. The rule will in other words always be put last in the ruleset in comparison to previously added rules, and hence be checked last, unless you append or insert more rules later on.
Command	**-D, --delete**
Example	**iptables -D INPUT --dport 80 -j DROP, iptables -D INPUT 1**
Explanation	This command deletes a rule in a chain. This could be done in two ways, either by specifying a rule to match with the **-D** option or by specifying the rule number that we want to match.
Command	**-R, --replace**
Example	**iptables -R INPUT 1 -s 192.168.0.1 -j DROP**
Explanation	This command replaces the old entry at the specified line. It works in the same way as the **--delete** command, but instead of totally deleting the entry, it will replace it with a new entry.
Command	**-I, --insert**
Example	**iptables -I INPUT 1 --dport 80 -j ACCEPT**
Explanation	Insert a rule somewhere in a chain. The rule is inserted at the actual number that we give. In other words, the above example would be inserted at place 1 in the INPUT chain, and hence it would be the absolutely first rule in the chain from now on.
Command	**-L, --list**
Example	**iptables -L INPUT**
Explanation	This command lists all the entries in the specified chain. In the above case, we would list all the entries in the INPUT chain. It's also legal to not specify any chain at all.
Command	**-F, --flush**
Example	**iptables -F INPUT**
Explanation	This command flushes the specified chain from all rules and is equivalent to deleting each rule one by one but is quite a bit faster. The command can be used without options, and will then delete all rules in all chains within the

	specified table.
Command	**-Z, --zero**
Example	**iptables -Z INPUT**
Explanation	This command tells the program to zero all counters in a specific chain or in all chains. If you have used the **-v** option with the **-L** command, you have probably seen the packet counter in the beginning of each field. To zero this packet counter, use the **-Z** option. This option works the same as **-L** except that **-Z** won't list the rules. If **-L** and **-Z** is used together (which is legal), the chains will first be listed, and then the packet counters are zeroised.
Command	**-N, --new-chain**
Example	**iptables -N allowed**
Explanation	This command tells the kernel to create a new chain by the specified name in the specified table. In the above example we create a chain called **allowed**. Note that there must be no target of the same name previously to creating it.
Command	**-X, --delete-chain**
Example	**iptables -X allowed**
Explanation	This command deletes the specified chain from the table. For this command to work there must be no rules that are referring to the chain that is to be deleted. In other words, you would have to replace or delete all rules referring to the chain before actually deleting the chain. If this command is used without any options, all chains that are not built in will be deleted from the specified table.
Command	**-P, --policy**
Example	**iptables -P INPUT DROP**
Explanation	This command tells the kernel to set a specified default target, or policy, on a chain. All packets that don't match any rule will then be forced to use the policy of the chain. Legal targets are: **DROP**, **ACCEPT** and **REJECT**
Command	**-E, --rename-chain**
Example	**iptables -E allowed disallowed**
Explanation	The **-E** command tells **iptables** to rename the first name of a chain, to the second name. In the example above we would, in other words, change the name of the chain from allowed to disallow.

A command should always be specified, unless you just want to list the built-in help for **iptables** or get the version of the command. To get the version, use the **-v** option and to get the help message, use the **-h** option.

7.3 Matches

Matches are of mainly five different categories First of all we have the *generic matches* which are generic and can be used in all rules. Then we have the *TCP matches* which can only be applied to TCP packets. We have *UDP matches* which can only be applied to UDP packets and *ICMP matches* which can only be used on ICMP packets. Finally we have special matches such as the state, owner and limit matches and so on. These final matches have in turn been split down to even more subcategories even though they might not necessarily be different matches at all.

7.3.1 Generic matches

A generic match is a kind of match that is always available whatever kind of protocol we are working on or whatever match extensions we have loaded. No special parameters are in other words needed to load these matches at all. The following matches are always available.

Generic matches

Match	**-p, --protocol**
Example	**iptables -A INPUT -p tcp**
Explanation	This match is used to check for certain protocols. Examples of protocols are TCP, UDP and ICMP. The command may also take a comma delimited list of protocols, such as **udp,tcp** which would match all UDP and TCP packets. If this match is given the numeric value of zero (0), it means ALL protocols, which in turn is the default behaviour in case the **--protocol** match is not used. This match can also be inversed with the **!** sign, so **--protocol ! tcp** would mean to match the ICMP and UDP protocols.
Match	**-s, --src, --source**
Example	**iptables –A INPUT -s 192.168.1.1**
Explanation	This is the source match which is used to match packets based on their source IP address. The main form can be used to match single IP addresses such as *192.168.1.1*. It could be used with a netmask in a bits form. One way is to do it with an regular netmask in the *255.255.255.255* form (ie, *192.168.0.0/255.255.255.0*), and the other way is to only specify the number of ones (1's) on the left side of the network mask. This means that we could for example add */24* to use a *255.255.255.0* netmask. We could then match whole IP ranges, such as our local networks or network segments behind the firewall. The line would then look something like, for example, *192.168.0.0/24*. This would match all packets in the *192.168.0.x* range. We could also inverse the match with an ! just as before. If we would in other words use a match in the form of **--source ! 192.168.0.0/24** we would match all packets with a source address not coming from within the *192.168.0.x* range. The default is to match all IP addresses.

Match	**-d, --dst, --destination**
Example	**iptables –A INPUT -d 192.168.1.1**
Explanation	The **--destination** match is used to match packets based on their destination address or addresses. It works pretty much the same as the **--source** match and has the same syntax, except that it matches based on where the packets are going. To match an IP range, we can add a net maskeither in the exact netmask form, or in the number of ones (1's) counted from the left side of the netmask bits. It would then look like either *192.168.0.0/255.255.255.0* or like *192.168.0.0/24* and both would be equivalent to each other. We could also invert the whole match with an **!** sign, just as before. **--destination ! 192.168.0.1** would in other words match all packets except those not destined to the *192.168.0.1* IP address.
Match	**-i, --in-interface**
Example	**iptables –A INPUT -i eth0**
Explanation	This match is used to match based on which interface the packet came in on. Note that this option is only legal in the INPUT, FORWARD and PREROUTING chains and will render an error message when used anywhere else. The default behavior of this match, in case the match is not specified, is to assume a string value of +. The + value is used to match a string of letters and numbers. A single + would in other words tell the kernel to match all packets without considering which interface it came in on. The + string can also be used at the end of an interface, and **eth+** would in other words match all Ethernet devices. We can also invert the meaning of this option with the help of the ! sign. The line would then have a syntax looking something like **-i ! eth0**, which would mean to match all incoming interfaces, except eth0.
Match	**-o, --out-interface**
Example	**iptables –A FORWARD -o eth0**
Explanation	The **--out-interface** match is used to match packets depending on which interface they are leaving on. Note that this match is only available in the OUTPUT, FORWARD and POSTROUTING chains, in opposite of the **--in-interface** match. Other than this, it works pretty much the same as the **--in-interface** match. The + extension is understood so you can match all eth devices with **eth+** and so on. To inverse the meaning of the match, you can use the **!** sign in exactly the same sense as in the **--in-interface** match. Of course, the default behavior if this match is left out is to match all devices, regardless of where the packet is going.
Match	**-f, --fragment**
Example	**iptables –A INPUT -f**
Explanation	This match is used to match the second and third part of a fragmented packet. The reason for this is that in the case of fragmented packets, there is no way to tell the source or destination ports of the fragments, nor ICMP

types, among other things. Also, fragmented packets might in rather special cases be used to compile attacks against computers. Such fragments of packets will not be matched by other rules when they look like this, and hence this match was created. This option can also be used in conjunction with the ! sign, however, in this case the ! sign must precede the match, like this ! **-f**. When this match is inversed, we match all head fragments and/or unfragmented packets. What this means is that we match all the first fragments of a fragmented packets, and not the second, third, and so on, fragments. We also match all packets that have not been fragmented during the transfer. Also note that there are defragmentation options within the kernel that can be used which are really good. In case connection tracking is used no defragmented packets are seen since they are dealt with before hitting any chain or table in **iptables**.

7.3.2 Implicit matches

Implicit matches are loaded automatically when we tell **iptables** that this rule will match for example TCP packets with the **--protocol** match. There are currently three types of *implicit matches* that are loaded automatically for three different protocols. These are *TCP matches*, *UDP matches* and *ICMP matches*. The TCP based matches contain a set of different matches that are available for only TCP packets, and UDP based matches contain another set of matches that are available only for UDP packets, and the same thing for ICMP packets.

TCP matches

These matches are protocol specific and are only available when working with TCP packets and streams. To use these matches you need to specify **--protocol tcp** on the command line before trying to use these matches. Note that the **--protocol tcp** match must be to the left of the protocol specific matches. These matches are loaded implicitly in a sense, just as the *UDP* and *ICMP matches* are loaded implicitly.

TCP matches

Match	**--sport, --source-port**
Example	**iptables -A INPUT -p tcp --sport 22**
Explanation	The **--source-port** match is used to match packets based on their source port. This match can either take a service name or a port number. If you specify a service name, the service name must be in the /etc/services file since **iptables** uses this file to look up the service name in. If you specify the port by port number, the entry of the rule will be slightly faster since **iptables** don't have to check up the service. The **--source-port** match can also be used to match a whole range of ports in this way **--source-port**

	22:80 .We could also invert a match by adding a **!** sign like **--source-port ! 22** which would mean that we want to match all ports but port 22.
Match	**--dport, --destination-port**
Example	**iptables -A INPUT -p tcp --dport 22**
Explanation	This match is used to match TCP packets depending on its destination port. It uses exactly the same syntax as the **--source-port** match. It understands port and port range specifications, as well as inversions. It does also reverse high and low ports in a port range specification if the high port went into the first spot and the low port into the last spot. The match will also assume the values of 0 or 65535 if the high or low port is left out in a port range specification. In other words, exactly the same as **--source-port** in syntax.
Match	**--tcp-flags**
Example	**iptables -p tcp --tcp-flags SYN,ACK,FIN SYN**
Explanation	This match is used to match depending on the TCP flags in a packet. The match knows about the SYN, ACK, FIN, RST, URG, PSH flags but it also recognizes the words ALL and NONE. ALL and NONE is pretty much self describing, ALL means to use all flags and NONE means to use no flags.
Match	**--syn**
Example	**iptables -p tcp –syn**
Explanation	The **--syn** is used to match packets if they have the SYN bit set and the ACK and FIN bits unset.
Match	**--tcp-option**
Example	**iptables -p tcp --tcp-option 16**
Explanation	This match is used to match packets depending on their TCP options. A TCP Option is a specific part of the header. This part consists of 3 different fields. The first one is 8 bits long and tells us which Options are used in this stream, the second one is also 8 bits long and tells us how long the options field is.

UDP matches

These matches only work with UDP packets. These matches are implicitly loaded when you specify the **--protocol UDP** match and will be available after this specification. Note that UDP packets are not connection oriented, and hence there is no such thing as different flags to set in the packet to give data on what the datagram is supposed to do, such as open or closing a connection, or if they are just simply supposed to send data. UDP packets do not require any kind of acknowledgement either.

UDP matches

Match	**--sport, --source-port**
Example	**iptables -A INPUT -p udp --sport 53**
Explanation	This match works exactly the same as its TCP counterpart. It is used to perform matches on packets based on their source UDP ports. It has support for port ranges, single ports and port inversions with the same syntax. To make a UDP port range you could do 22:80 which would match UDP ports 22 through 80. If the first value is omitted, port 0 is assumed. If the last port is omitted, port 65535 is assumed. To invert the port match, add a **!** sign in this, **--source-port ! 53** fashion.
Match	**--dport, --destination-port**
Example	**iptables -A INPUT -p udp --dport 53**
Explanation	The same goes for this match as for the UDP version of **--source-port**, it is exactly the same as the equivalent TCP match, but will work with UDP packets instead. The match is used to match packets based on their UDP destination port. The match handles port ranges, single ports and inversions.

ICMP matches

These are the *ICMP matches*. These packets are even worse than UDP packets in the sense that they are connectionless. The ICMP protocol is mainly used for error reporting and for connection controlling and such features. ICMP is not a protocol subordinated to the IP protocol, but more of a protocol beside the IP protocol that helps handling errors. The headers of a ICMP packet are very similar to those of the IP headers, but contains differences. The main feature of this protocol is the type header which tells us what the packet is to do. There is only one ICMP specific match available for ICMP packets. This match is implicitly loaded when we use the **--protocol ICMP** match and we get access to it automatically.

ICMP matches

Match	**--icmp-type**
Example	**iptables -A INPUT -p icmp --icmp-type 8**
Explanation	This match is used to specify the ICMP type to match. ICMP types can be specified either by their numeric values or by their names. Numerical values are specified in RFC 792. This match can also be inverted with the **!** sign in this, **--icmp-type ! 8.**

7.3.3 Explicit matches

Explicit matches are matches that must be specifically loaded with the **-m** or **--match** option. If we would like to use the state matches for example, we would have to write **-m**

state to the left of the actual match using the state matches. The difference between implicitly loaded matches and explicitly loaded ones is that the implicitly loaded matches will automatically be loaded when you, for example, match TCP packets, while explicitly loaded matches will not be loaded automatically in any case and it is up to you to activate them before using them.

MAC match

The MAC match can be used to match packets based on their MAC source address. This match is a little bit limited, however, in the future this may be more evolved and may be more useful.

MAC match options

Match	**--mac-source**
Example	**iptables -A INPUT -m mac --mac-source 00:00:00:00:00:01**
Explanation	This match is used to match packets based on their MAC source address. The MAC address specified must be in the form *XX:XX:XX:XX:XX:XX*, else it will not be legal. The match may be reversed with an **!** sign and would look like **--mac-source ! 00:00:00:00:00:01**. Since MAC addresses are only used on ethernet type networks, this match will only be possible to use on ethernet based networks. This match is also only valid in the PREROUTING, FORWARD and INPUT chains and nowhere else.

Limit match

The **limit** match extension must be loaded explicitly with the **-m limit** option. This match is excellent to use to do limited logging of specific rules etcetera. For example, you could use this to match all packets that goes over the edge of a certain chain, and get limited logging of this. What this means, is that when we add this match we **limit** how many times a certain rule may be matched in a certain timeframe. This is its main usage, but there are more usages, of course.

Limit match options

Match	**--limit**
Example	**iptables -A INPUT -m limit --limit 3/hour**
Explanation	This sets the maximum average matching rate of the **limit** match. This match is specified with a number and an optional time specifier. The following time specifiers are currently recognised: **/second /minute /hour /day**. The default value here is 3 per hour, or 3/hour. This tells the **limit** match how many times to let this match run per timeunit (ie /minute).

Match	**--limit-burst**
Example	**iptables -A INPUT -m limit --limit-burst 5**
Explanation	This is the setting for the *burst limit* of the **limit** match. It tells **iptables** the maximum initial number of packets to match. This number gets recharged by one every time the average limit specified (with the **--limit** option) is not reached, up to the number specified with the **--limit-burst** option. When and if the burst limit is reached, we go down to the lowest possible delimiter, 1. After this, we get one "token" for every timeround specified that we do not hit the new delimiter, until the delimiter reach the burst limit again.. The default **--limit-burst** value is 5.

Multiport match

The **multiport** match extension can be used to specify more destination ports and port ranges than one, which would sometimes mean you would have to make several rules looking exactly the same just to match different ports.

Multiport match options

Match	**--source-port**
Example	**iptables -A INPUT -p tcp -m multiport --source-port 22,53,80,110**
Explanation	This match matches multiple source ports. A maximum of 15 separate ports may be specified. The ports must be comma delimited, as you can see in the example. This match may only be used in conjunction with the **-p tcp** or **-p udp** matches. It is mainly an enhanced version of the normal **--source-port** match.
Match	**--destination-port**
Example	**iptables -A INPUT -p tcp -m multiport --destination-port 22,53,80,110**
Explanation	This match is used to match multiple destination ports. It works exactly the same way as the source port match mentioned just above, except that it matches destination ports. It has a maximum specification of 15 ports and may only be used in conjunction with **-p tcp** and **-p udp**.
Match	**--port**
Example	**iptables -A INPUT -p tcp -m multiport --port 22,53,80,110**
Explanation	This match extension can be used to match packets based both on their destination port and their source port. It works the same way as the **--source-port** and **--destination-port** matches above. It can take a maximum of 15 ports specified to it in one argument. It can only be used in conjunction with **-p tcp** and **-p udp**. Note that this means that it will only match packets that comes from, for example, port 80 to port 80 and if you have specified port 80 to the **--port** match.

Mark match

The **mark** match extension is used to match packets based on the marks they have set. A **mark** is a special field only maintained within the kernel that is associated with the packets as they travel through the computer. They may be used by different kernel routines for such tasks as traffic shaping and filtering. As of today, there is only one way of setting a mark in Linux, namely the **MARK** target in **iptables**. This was previously done with the **FWMARK** target in **ipchains**. The mark field is currently set to an unsigned integer, or 4294967296 possible values on a 32 bit system.

Mark match options

Match	**--mark**
Example	**iptables -t mangle -A INPUT -m mark --mark 1**
Explanation	This match is used to match packets that have previously been marked. All packets traveling through netfilter gets a special mark field associated with them. This mark field does not in any way travel outside, with or without the packet, the actual computer itself. If this mark field matches the **mark** match it is a match. The mark field is an unsigned integer, hence there can be a maximum of 65535 different marks.

Owner match

The **owner** match extension is used to match packets based on who created them. This extension was originally written as an example on what **iptables** might be used for. This match only works within the OUTPUT chain as it looks today, for obvious reasons. It is pretty much impossible to find out any information about who sent a packet on the other end, or if we where an intermediate hop to the real destination. Even within the OUTPUT chain it is not very reliable since certain packets may not have an owner. Notorious packets of that sort is different ICMP responses among other things. ICMP responses will, hence, never match.

Owner match options

Match	**--uid-owner**
Example	**iptables -A OUTPUT -m owner --uid-owner 500**
Explanation	This packet match will match if the packet was created by the given User ID (UID). This could be used to match outgoing packets based on who created them.
Match	**--gid-owner**
Example	**iptables -A OUTPUT -m owner --gid-owner 0**

Explanation	This match is used to match all packets based on their Group ID (GID). This means that we match all packets based on what group the user creating the packets are in.
Match	**--pid-owner**
Example	**iptables -A OUTPUT -m owner --pid-owner 78**
Explanation	This match is used to match packets based on their Process ID (PID) and which PID created the packets.
Match	**--sid-owner**
Example	**iptables -A OUTPUT -m owner --sid-owner 100**
Explanation	This match is used to match packets based on their Session ID and the Session ID used by the program in question. The SID, or Session ID, value is set upon different processes depending on their originating process if they are threaded, or upon which process created it.

State match

The **state** match extension is used in conjunction with the connection tracking code in the kernel and allows access to the connection tracking state of the packets. This allows us to know in what state the connection is, and works for pretty much all protocols, including stateless protocols such as ICMP and UDP. In all cases, there will be a default timeout for the connection and it will then be dropped from the connection tracking database. This match needs to be loaded explicitly by adding a **-m state** statement to the rule.

State matches

Match	**--state**
Example	**iptables -A INPUT -m state --state RELATED,ESTABLISHED**
Explanation	This match option tells the **state** match what states the packets must be in to be matched. There is currently 4 states that can be used. **INVALID**, **ESTABLISHED**, **NEW** and **RELATED**.

Unclean match

The **unclean** match takes no options and requires no more than explicit loading when you want to use it.This option is regarded as experimental and may not work at all times, nor will it take care of all unclean packages or problems. This match tries to match packets which seems malformed or unusual, such as packets with bad headers or checksums and so on. This could be used to **DROP** connections and to check for bad streams etc.

TOS match

The **TOS** match can be used to match packets based on their TOS field. TOS stands for Type Of Service, consists of 8 bits, and is located in the IP header. This match is loaded explicitly by adding **-m tos** to the rule. TOS is normally used to tell intermediate hosts the preceeding of the stream, and what kind of content it has(not really, but it tells us if there is any specific requirements for this stream such as that it needs to be sent as fast as possible, or it needs to be able to send as much payload as possible

TOS matches

Match	**--tos**
Example	**iptables -A INPUT -p tcp -m tos --tos 0x16**
Explanation	This match is used as described above, it can match packets based on their TOS field and their value. This could be used for, among other things, to **mark** packets for later usage together with the **iproute2** and advanced routing functions in linux. The match takes an hex or numeric value as an option, or possibly one of the names given if you do an **iptables -m tos -h**.

TTL match

The **TTL** match is used to match packets based on their TTL (Time To Live) field residing in the IP header. The TTL field contains 8 bits of data and is decremented once every time it is processed by an intermediate host between the client and host. If the TTL reaches 0, an ICMP type 11 code 0 (TTL equals 0 during transit) or code 1 (TTL equals 0 during reassembly) is transmitted to the party sending the packet and telling about the problem. This match is only used to match packets based on their TTL, and not to change anything. This is true here, as well as in all kinds of matches. To load this match, you need to add an **-m ttl** to the rule.

TTL matches

Match	**--ttl**
Example	**iptables -A OUTPUT -m ttl --ttl 60**
Explanation	This match option is used to specify which TTL value to match. It takes an numeric value and matches based on this value.

7.4 Targets/Jumps

The target/jumps tell the rule what to do with a packet that is a perfect match with the match section of the rule. There is a few basic targets, the **ACCEPT** and **DROP** targets which we will deal with first of all targets.

The jump specification is done exactly the same as the target definition except that it requires a chain within the same table to jump to. To jump to a specific chain, it is required that the chain has already been created. As we have already explained before, a chain is created with the **-N** command. For example, let's say we create a chain in the filter table called **tcp_packets** like this: **iptables -N tcp_packets**. We could then add a jump target to it like this: **iptables -A INPUT -p tcp -j tcp_packets**. We would then jump from the **INPUT** chain to the **tcp_packets** chain and start traversing that chain. When/If we reach the end of that chain, we get dropped back to the **INPUT** chain and the packet starts traversing from the rule one step below where it jumped to the other chain (tcp_packets in this case). If a packet is **ACCEPT**'ed within one of the subchains, it will automatically be **ACCEPT**'ed in the superset chain also and it will not traverse any of the superset chains any further. However, the packet will traverse all other chains in the other tables in a normal fashion.

Targets on the other hand specify an action to take on the packet in question. We could for example, **DROP** or **ACCEPT** the packet depending on what we want to do. There is also a number of other actions we may want to take which we will describe further on in this section. Targets may also end with different results one could say, some targets will make the packet stop traversing the specific chain and superset chains as described above. Good examples of such rules are **DROP** and **ACCEPT**. Rules that are stopped, will not pass through any of the rules further on in the chain or superset chains. Other targets, may take an action on the packet and then the packet will continue passing through the rest of the rules anyway, a good example of this would be the **LOG**, **DNAT** and **SNAT** targets. These packets may be logged, Network Address Translationed and then be passed on to the other rules in the same chains. This may be good in cases where we want to take two actions on the same packet, such as both mangling the TTL and the TOS value of a specific packet/stream. Some targets will also take options that may be necessary (What address to do NAT to, what TOS value to use etc) while others have options not necessary, but available in any case (log prefixes, masquerade to ports and so on).

7.4.1 ACCEPT target

This target takes no special options first of all. When a packet is perfectly matched and this target is set, it is accepted and will not continue traversing the chain where it was accepted in, nor any of the calling chains.Packets that was accepted in one chain will still travel through any subsequent chains within the other tables and may be dropped there. There is nothing special about this target whatsoever, and it does not require, or have the possibility, to add options to the target. To use this target, we specify it like **-j ACCEPT**.

7.4.2 DROP target

The **DROP** target does just what it says, it drops packets and refuses to process them anymore. A packet that matches a rule perfectly and then has this action taken on it will be blocked and no further processing will be done.This action may be a bit bad in certain cases since it may leave dead sockets around on the server and client. A better solution would be to use the **REJECT** target in those cases, especially when you want to block certain portscanners from getting to much information, such as filtered ports and so on. Also that if a packet has the **DROP** action taken on them in a subchain, the packet will not be processed in any of the above chains in the structure, nor by any other tables. The packet is in other words totally dead. The target will not send any kind of information in either direction, either to tell the client or the server as told previously.

7.4.3 QUEUE target

The QUEUE target is used to queue packets to userland programs and applications. This is done in a programmatic way and may be used to, for example, create network accounting or to create specific and advanced applications to proxy or filter packets.

7.4.4 RETURN target

The **RETURN** target will make the current packet stop traveling through the chain where it hit the rule. If it is a subchain to another chain, the packet will continue to travel through the above chains in the structure as if nothing had happened. If the chain is the main chain, for example the INPUT chain, the packet will have the default policy taken on it. The default policy is normally set to **ACCEPT** or **DROP** or something the like.

7.4.5 LOG target

The **LOG** target is specially made to make it possible to log snippets of information about packets that may be illegal, or for pure bug hunting and error finding. The **LOG** target will log specific information such as most of the IP headers and other interesting information via the kernel logging facility. This information may then be read with **dmesg** or **syslogd** and likely programs and applications.

7.4.6 MARK target

The **MARK** target is used to set **netfilter** mark values that are associated with specific packets. This target is only valid in the mangle table, and will not work outside there.

7.4.7 REJECT target

The **REJECT** target works basically the same as the **DROP** target, but it also sends back an error message to the host sending the packet that was blocked. The **REJECT** target is as of today only valid in the INPUT, FORWARD and OUTPUT chain or subchains of

those chains, which would also be the only chains where it would make any sense to put this target in.

7.4.8 TOS target

The **TOS** target is used to set the Type of Service field within the IP header. The TOS field consists of 8 bits which are used to route packets. This is one of the fields that can be used directly within **iproute2** and its subsystem to route packets.

7.4.9 MIRROR target

The **MIRROR** target is an experimental demonstration target only, and you should be warned of using this since it may result in really bad loops, hence resulting in a bad kind of Denial of Service, among other things. The **MIRROR** target is used to invert the source and destination fields in the IP header, and then to retransmit the packet

7.4.9 SNAT target

The **SNAT** target is used to do Source Network Address Translation, which means that this target will rewrite the Source IP address in the IP header of the packet. For example, this is good when we want several computers to share an internet connection. We could then turn on ip forwarding in the kernel, and then set an **SNAT** rule which would translate all packets from our local network to the **source IP** of our own internet connection. Without doing this, the outside world would not know where to send reply packets, since our local networks should use the IANA specified IP addresses which are allocated for **LAN** networks. If we forwarded these packets as is, none on the internet would know that they where actually from us. The **SNAT** target does all the translation needed to do this kind of work, letting all packets leaving our **LAN** look as if they came from a single host, which would be our firewall.

The **SNAT** target is only valid within the nat table, within the POSTROUTING chain. This is in other words the only place that you may do **SNAT** in. If the first packet in a connection is mangled in this fashion, then all future packets in the same connection will also be **SNAT**'ed and, also, no further processing of rules in the POSTROUTING chain will be commenced on the packets in the same stream.

SNAT target

Option	**--to-source**
Example	**iptables -t nat -A POSTROUTING -o eth0 -j SNAT --to-source 194.236.50.155-194.236.50.160:1024-32000**
Explanation	The **--to-source** option is used to specify which source the packets should use. This option, at it simplest, takes one IP address to which we should transform all the **source IP** addresses in the **IP header**. If we want to balance between several IP addresses we could use an range of IP addresses

separated by a hyphen, it would then look like, for example, 194.236.50.155-194.236.50.160 as described in the example above. The source IP would then be set randomly for each stream that we open, and a single stream would always use the same IP address for packets within that stream. There may also be an range of ports specified that should only be used by **SNAT**. All the source ports would then be mapped to the ports specified. This would hence look as within the example above, :1024-32000 or something alike. iptables will always try to not make any port alterations if it is possible, but if two hosts tries to use the same ports, iptables will map one of them to another port. If no port range is specified, then all source ports below 512 will be mapped to other ports below 512 if needed. Those between source ports 512 and 1023 will be mapped to ports below 1024. All other ports will be mapped to 1024 or above. As previously stated, iptables will always try to maintain the source ports used by the actual workstation making the connection. Note that this has nothing to do with destination ports, so if a client tries to make contact with an **HTTP** server outside the firewall, it will not be mapped to the **FTP control** port.

7.4.10 DNAT target

The **DNAT** target is used to do Destination Network Address Translation, which means that it is used to rewrite the Destination IP address of a packet. If a packet is matched, and this is the target of the rule, the packet, and all subsequent packets in the same stream will be translated, and then routed on to the correct device, host or network. This target can be extremely useful, for example, when you have an host running your webserver inside a *LAN*, but no real IP to give it that will work on the internet. You could then tell the firewall to forward all packets going to its own HTTP port, on to the real webserver within the *LAN*. We may also specify a whole range of destination IP addresses, and the **DNAT** mechanism will choose the destination IP address at random for each stream. Hence, we will be able to deal with a kind of load balancing by doing this.

The **DNAT** target is only available within the PREROUTING and OUTPUT chains in the nat table, and any of the chains called upon from any of those listed chains. Chains containing **DNAT** targets may not be used from any other chains, such as the POSTROUTING chain.

DNAT target

Option	**--to-destination**
Example	**iptables -t nat -A PREROUTING -p tcp -d 15.45.23.67 --dport 80 -j DNAT --to-destination 192.168.1.1-192.168.1.10**
Explanation	The **--to-destination** option tells the DNAT mechanism which Destination IP to set in the IP header, and where to send packets that are matched. The

above example would send on all packets destined for IP address 15.45.23.67 to a range of *LAN* IP's, namely 192.168.1.1 through 10. A single stream will always use the same host, and each stream will randomly be given an IP address that it will always be Destinated for, within that stream. We could also have specified only one IP address, in which case we would always be connected to the same host. We may add an port or port range to which the traffic would be redirected to. This is done by adding, for example, an :80 statement to the IP addresses to which we want to DNAT the packets. A rule could then look like **--to-destination 192.168.1.1:80** for example, or like **--to-destination 192.168.1.1:80-100** if we wanted to specify a port range.

7.4.11 MASQUERADE target

The **MASQUERADE** target is used basically the same as the **SNAT** target, but it does not require any **--to-source** option. The reason for this is that the **MASQUERADE** target was made to work with, for example, dialup connections, or DHCP connections, which gets dynamic IP addresses when connecting to the network in question. This means that **MASQUERADE** target should only be used with dynamically assigned IP connections, whose actual address is not known at all times.

When a connection is masqueraded, it means that we set the IP address used on a specific network interface instead of the **--to-source** option, and the IP address is automatically grabbed from the information about the specific interface.

MASQUERADE target is only valid within the POSTROUTING chain in the nat table, just as the **SNAT** target. The **MASQUERADE** target takes one option specified below, which is optional.

MASQUERADE target

Option	**--to-ports**
Example	**iptables -t nat -A POSTROUTING -p TCP -j MASQUERADE --to-ports 1024-31000**
Explanation	The **--to-ports** option is used to set the source port or ports to use on outgoing packets. Either you can specify a single port like **--to-ports 1025** or you may specify a port range as **--to-ports 1024-3000**. The **--to-ports** option is only valid if the rule match section specifies the TCP or UDP protocols with the **--protocol** match.

7.4.12 REDIRECT target

The **REDIRECT** target is used to redirect packets and streams to the machine itself. This means that we could for example **REDIRECT** all packets destined for the HTTP ports to an HTTP proxy like squid, on our own host. Locally generated packets are mapped to the 127.0.0.1 address. The **REDIRECT** target is extremely good to use when we want, for example, transparent proxying, where the *LAN* hosts do not know about the proxy at all.

REDIRECT target is only valid within the PREROUTING and OUTPUT chains of the nat table. It is also valid within user-defined chains that are only called from those chains, and nowhere else. The **REDIRECT** target takes only one option, as described below.

REDIRECT target

Option	**--to-ports**
Example	**iptables -t nat -A PREROUTING -p tcp --dport 80 -j REDIRECT --to-ports 8080**
Explanation	The **--to-ports** option specifies the destination port, or port range, to use. Without the **--to-ports** option, the destination port is never altered. This is specified, as above, **--to-ports 8080** in case we only want to specify one port. If we would want to specify an port range, we would do it like **--to-ports 8080-8090**, which tells the **REDIRECT** target to redirect the packets to the ports 8080 through 8090.

7.4.13 TTL target

The **TTL** target is used to modify the Time To Live field in the IP header. One useful application of this is to change all Time To Live values to the same value on all outgoing packets.

The **TTL** target is only valid within the mangle table, and nowhere else. It takes 3 options, all of them described below in the table.

TTL target

Option	**--ttl-set**
Example	**iptables -t mangle -A PREROUTING -i eth0 -j TTL --ttl-set 64**
Explanation	The **--ttl-set** option tells the **TTL** target which TTL value to set on the packet in question.
Option	**--ttl-dec**
Example	**iptables -t mangle -A PREROUTING -i eth0 -j TTL --ttl-dec 1**

Explanation The **--ttl-dec** option tells the **TTL** target to decrement the Time To Live value by the amount specified after the **--ttl-dec** option.

Option **--ttl-inc**

Example **iptables -t mangle -A PREROUTING -i eth0 -j TTL --ttl-inc 1**

Explanation The **--ttl-inc** option tells the **TTL** target to increment the Time To Live value with the value specified to the **--ttl-inc** option.

7.4.14 ULOG target

The **ULOG** target is used to provide userspace logging of matching packets. If a packet is matched and the **ULOG** target is set, the packet information is multicasted together with the whole packet through a netlink socket. One or more userspace processes may then subscribe to various multicast groups and receive the packet.

Chapter 8

Example Implementation at Computer Center Q.A.U

Our firewall serves two purposes. Its primary function is that of a firewall and security monitor for our secure subnet. This allows us to monitor all incoming and outgoing traffic. Neither side of the firewall has any direct connection to the other except through the firewall host. This meets the definition of a BastionHost. The second purpose is to act as a Router between the outside network and the internal network.

8.1 System Selection

Selection of the hardware and software components of the firewall is critical. A limited capacity machine will lead to network congestion and eventual collapse of the firewall due to overloading.

8.1.1 Hardware Requirements

Filtering firewalls don't require much hardware. They are little more than simple routers. All that was used is as follows:

1. A Pentium III with 256 Mb of RAM
2. A 20 Giga byte Hard disk.
3. Two Network Interface Cards
4. Monitor and keyboard

8.1.2 Software requirements

The Distribution chosen for the firewall was Red Hat Linux 7.3. This package was chosen for a variety of reasons. These reasons include:

1. Ease of installation.
2. Stability of the Kernel
3. Utilities included with the distribution.
4. Final installation configuration.
5. Kernel 2.4.18 (with precompiled support for Iptables)

8.2 Preparing the Linux system

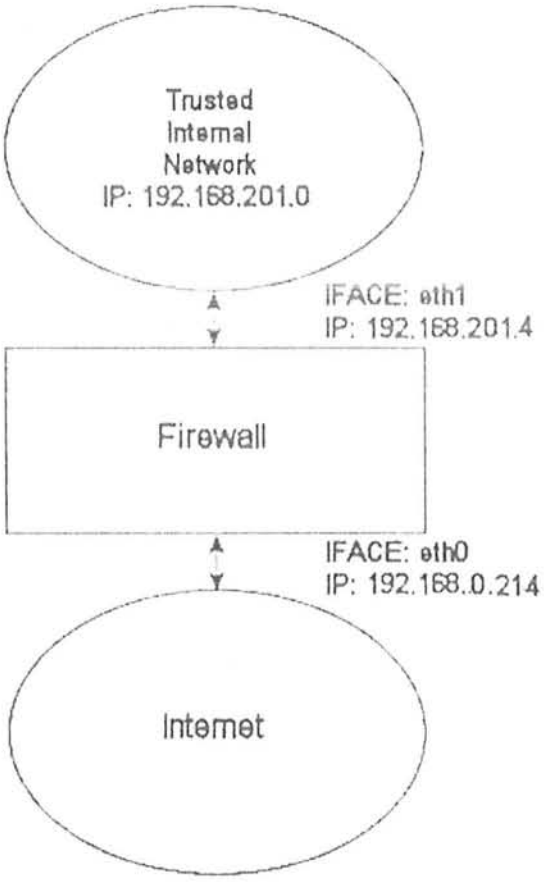
Once the system has been selected, it must be prepared for use as a firewall by making numerous changes both to the system software and the hardware attached to the computer

system. Installation was done with as little services as possible. My installation started with a server configuration having 512 MB of **"/swap"** and 50 MB **"/boot"** partition then I turn off ever un-needed service in /etc/xinetd.conf.
Because the distribution came with a kernel usefull to my purpose. There was no need to compile my own kernal.

8.3 Configuring two network cards

My goal here is to provide two network connection to my filtering firewall system. One on the Internet (unsecured side) and one on the LAN (secure side). NIC were configured using network configuration tool.

192.168.0.214. is the outside connection to internet that I am using so I will use 192.168.201.xxx in my LAN. I assigned IP number 192.168.201.4 to the LAN card. This is our gateway IP address. We can assign all the other machines in the protected network (LAN) a number in the 192.168.201.xxx range. (192.168.201.1 through 192.168.201.254)



8.4 Testing the network

I started by using the ifconfig and route commands. With two network cards ifconfig looked something like:

#ifconfig

```
lo      Link encap:Local Loopback
        inet addr:127.0.0.1  Mask:255.0.0.0
        UP LOOPBACK RUNNING  MTU:3924  Metric:1
        RX packets:1620 errors:0 dropped:0 overruns:0
        TX packets:1620 errors:0 dropped:0 overruns:0
        collisions:0 txqueuelan:0

eth0    Link encap:10Mbps Ethernet  HWaddr 00:00:09:85:AC:55
        inet addr:192.168.0.214 Bcast:192.168.0.255  Mask:255.255.255.0
        UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
        RX packets:1000 errors:0 dropped:0 overruns:0
        TX packets:1100 errors:0 dropped:0 overruns:0
        collisions:0 txqueuelan:0
        Interrupt:12 Base address:0x310

eth1    Link encap:10Mbps Ethernet  HWaddr 00:00:09:80:1E:D7
        inet addr:192.168.201.1  Bcast:192.168.201.255  Mask:255.255.255.0
        UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
        RX packets:1110 errors:0 dropped:0 overruns:0
        TX packets:1111 errors:0 dropped:0 overruns:0
        collisions:0 txqueuelan:0
        Interrupt:15 Base address:0x350
```

and route table looks like this:

#route -n

Kernel routing table

Destination	Gateway	Genmask	Flags	MSS	Window	Use	Iface
192.168.0.0	*	255.255.255.0	U	1500	0	15	eth0
192.168.201.0	*	255.255.255.0	U	1500	0	0	eth1
127.0.0.0	*	255.0.0.0	U	3584	0	2	lo
default	192.168.0.10	*	UG	1500	0	72	eth0

Note: 192.168.0.0 is the Internet side of this firewall and 192.168.201.0 is the private (LAN) side.

Now I have to be sure that every computer on my LAN can ping the inside address of the firewall system. (192.168.201.1)

Now by turning on IP forwarding. I was able to ping anywhere on the Internet from any system on the LAN.

echo "1" > /proc/sys/net/ipv4/ip_forward

8.5 Initial System security

A firewall isn't any good if the system it is build on is left wide open to attacks. A "hacker" could gain access through a non firewall service and modify it for their own needs.

8.5.1 BIOS/CMOS Settings

In the CMOS setup BIOS password was set and all boot devices except Hard Disk, were disabled.

8.5.2 /tmp

It was ensured that only the file's owner can delete a given file in /tmp by giving following command:

chmod 1777 /tmp

8.5.3 Control-Alt-Delete keyboard shutdown command

This is important because server don't have the best physical security. It was done by editing **"/etc/inittab"** and changing the line

ca::ctrlaltdel:/sbin/shutdown -t3 -r now
to
#ca::ctrlaltdel:/sbin/shutdown -t3 -r now

then for the system to understand the change, following command was given

/sbin/init q

8.5.4 Disabling the ability to run INIT in interactive mode

It was done by editing **"/etc/sysconfig/init "**script and changing the line
prompt=yes
to..
prompt=no

8.5.5 Change what system daemons get loaded by editing the following files in "/etc/rc.d/"

The way that Redhat boots is the SysV way. This is where the OS will execute ALL files for a given runlevel

A run-level is the mode that the machine will load various system programs. Though this varies from Unix to Unix (Linux, Solaris, AIX, HP-UX, etc.), they are similar. For Linux, this is the run-levels (from /etc/inittab):

- 0: halt (stops the OS and sometimes shuts the power off)
- 1: single user (doesn't bring up the network, no passwd for root. Needed for system problems, lost root passwds, etc)
- 2: Multiuser (Brings up the whole OS but doesn't mount remote file systems (NFS, CODA, etc)
- 3: Full Multiuser (Brings up the whole OS with any remote file systems) 4: Unused
- 5: X-windows (Brings up the system immediately into X-windows)
- 6: Reboot (reboots the machine; usually into a COLD boot state [counts all the RAM, etc])

All of the files in various runlevel directories like /etc/rc.d/rc0, 1, 2, 3, 4, 5, 6.d are actually just symbolic links to all the real script files in /etc/rc.d/init.d. This makes things more manageable.So, since Linux usually runs in multi-user / non-Xwindows mode, that means runlevel "3" will execute all files in the /etc/rc.d/rc3.d directory.

8.5.6 Shutting down most of inetd / xinetd

Inetd and Xinetd are called the "super servers" as they load a network server based upon a request from the network.

Both of the runlevel configurations and removal of not needed daemons was done using a Gnome GUI tool **"serviceconfig".**

8.5.7 Shadow Passwords

In most early Linux distributions, all user's passwords were stored in the /etc/passwd file. These passwords were then encrypted by the "crypt" tool. The problem with this setup was that anyone could get these encrypted passwords and crypt's encryption was very poor. These passwords could then be broken with publically available tools. In recent times, the shadow system was implemented where the passwords were hashed with the MD5 algorithm and placed the resulting MD5 hased passwords in /etc/shadow.

It was confirmed that system is using SHADOW passwords by looking at the /etc/passwd file and

8.5.8 Disabling miscellaneous cron stuff

When Redhat is installed, usually more programs than planned are installed. Though Redhat allows users to later choose what daemons are and are NOT run upon boot, this does NOT disable some things that are loaded into the cron file.
By looking in the **"/etc/cron.hourly"**, **" /etc/cron.daily"**, **"/etc/cron.weekly"**, and **"/etc/cron.monthly"** it was made sure that nothing is installed that is not needed.

8.6 Firewalling Script

The script requires the following options to be compiled statically to the kernel, or as modules. Without one or more of these, the script will become more or less flawed since parts of the scripts required functionalities will be unusable. As the script can be changed, than more options may be needed to be compiled into the kernel depending on the requirement.

- CONFIG_PACKET
- CONFIG_PACKET
- CONFIG_NETFILTER
- CONFIG_IP_NF_CONNTRACK
- CONFIG_IP_NF_IPTABLES
- CONFIG_IP_NF_MATCH_LIMIT
- CONFIG_IP_NF_MATCH_STATE
- CONFIG_IP_NF_FILTER
- CONFIG_IP_NF_NAT
- CONFIG_IP_NF_TARGET_LOG

Now that every thing is setup and system is ready we can run our firewall script.
The main parts of the script are explained briefly below:

8.6.1 Configuration options

The first section within the script is the configuration section. The **$INET_IP** should always be a fully valid IP address. Also, the **$INET_IFACE** variable should point to the actual device used for internet connection. This could be eth0, eth1, ppp0 etc

Second section only consists of the **$IPTABLES** variable, which will point the script to the exact location of the **iptables** application. The default location when compiling the iptables package by hand is "`/usr/local/sbin/iptables`". However, many distributions put the actual application in another location such as "`/sbin/iptables`" and so on.

8.6.2 proc set up

At this point we start the IP forwarding by echoing a 1 to `/proc/sys/net/ipv4/ip_forward` in this fashion:

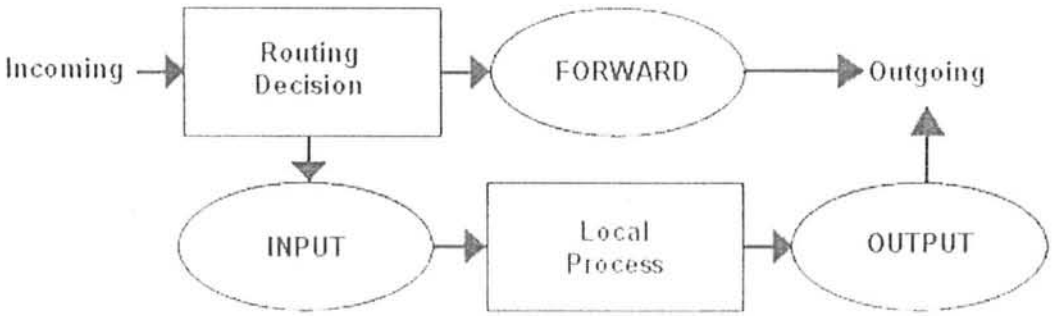
echo "1" > /proc/sys/net/ipv4/ip_forward

and also do other /proc settings.

8.6.3 Displacement of rules to different chains

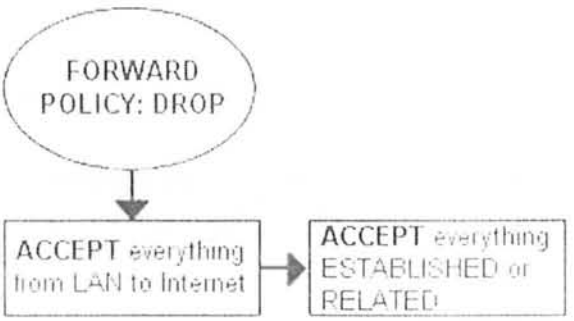
In this section I will briefly describe my choices within the script regarding user specified chains and some choices specific to the script.

I have displaced all the different user-chains in the way to save as much CPU time as possible but at the same time put the main weight on security and readability. Instead of letting a TCP packet traverse ICMP, UDP and TCP rules, I simply match all TCP packets and then let the TCP packets traverse an user specified chain. This way we do not get too much overhead out of it all. The following picture will try to explain the basics of how an incoming packet traverses netfilter. With these pictures and explanations, I wish to explain and clarify the goals of this script.

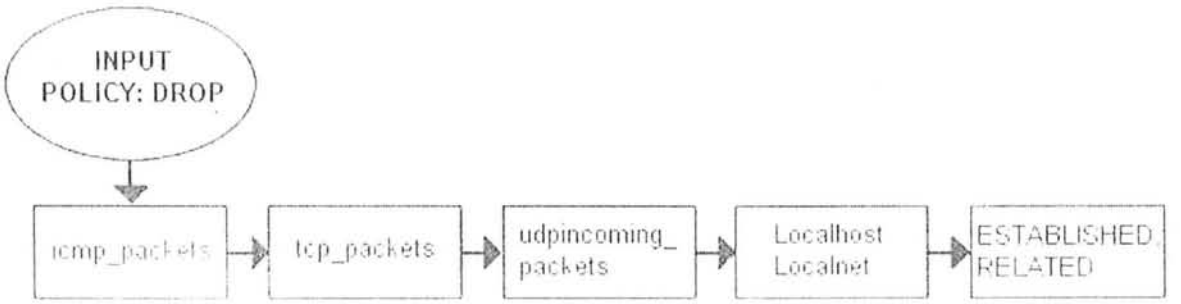


This whole script is based upon the assumption that we have one local network, one firewall and an Internet connection connected to the firewall. This is also based upon the assumption that we have a static IP to the internet. We also want to allow the firewall to act as a server for certain services on the internet, and we trust our local network fully and hence we will not block any of the traffic from the local network. Also, this script has as a main priority to only allow traffic that we explicitly want to allow. To do this, we want to set default policies within the chains to DROP. This will effectively kill all connections and all packets that we do not explicitly allow inside our network or our firewall.

We would also like to let our local network do connections to the internet. Since the local network is fully trusted, we want to allow all kind of traffic from the local network to the internet. However, the Internet is most definitely not a trusted network and hence we want to block them from getting to our local network. Based upon these general assumptions, let's look at what we need to do and what we do not need and want to do.



First of all, we want the local network to be able to connect to the internet, of course. To do this, we will need to NAT all packets since none of the local computers have real IP addresses. All of this is done within the PREROUTING chain, which is created last in this script. This means that we will also have to do some filtering within the FORWARD chain since we will otherwise allow outsiders full access to our local network. We trust our local network to the fullest, and because of that we specifically allow all traffic from our local network to the internet. Since no one on the Internet should be allowed to contact our local network computers, we will want to block all traffic from the Internet to our local network except already established and related connections, which in turn will allow all return traffic from the Internet to our local network.



As there is no service provided by internal network to the internet. Therefore, I have decided to allow HTTP, FTP & SSH access to the actual firewall. All of these protocols are available on the actual firewall, and hence it should be allowed through the INPUT chain, and we need to allow the return traffic through the OUTPUT chain. However, we also trust the local network fully, and the loopback device and IP address are also trusted. Because of this, we want to add special rules to allow all traffic from the local network as well as the loopback network interface. Also, we do not want to allow specific packets or packet headers in specific conjunctions, nor do we want to allow some IP ranges to reach the firewall from the Internet. For instance, the 10.0.0.0/8 address range is reserved for local networks and hence we would normally not want to allow packets from such a address range since they would certainly be spoofed.

Since we have an FTP server running on the server, as well as the fact we want to traverse as few rules as possible, we add a rule which lets all established and related traffic through at the top of the INPUT chain. For the same reason, we want to split the rules down into subchains. By doing this, our packets will hopefully only need to traverse as few rules as possible. By traversing less rules, we make the ruleset less timeconsuming for each packet, and reduce redundancy within the network.

In this script, we choose to split the different packets down by their protocol family, for example TCP, UDP or ICMP. All TCP packets traverse a specific chain named **"tcp_packets"**, which will contain rules for all TCP ports and protocols that we want to allow. Also, we want to do some extra checking on the TCP packets, so we would like to create one more subchain for all packets that are accepted for using valid port numbers to the firewall. This chain we choose to call the allowed chain, and should contain a few extra checks before finally accepting the packet. As for ICMP packets, these will traverse the **"icmp_packets"** chain. Finally, we have the UDP packets which needs to be dealt

with. These packets, we send to the "**udp_packets**" chain which handles all incoming UDP packets. All incoming UDP packets should be sent to this chain, and if they are of an allowed type we should accept them immediately without any further checking.

Finally, we have the firewalls OUTPUT chain. Since we actually trust the firewall quite a lot, we allow pretty much all traffic leaving the firewall. We do not do any specific user blocking, nor do we do any blocking of specific protocols. However, we do not want people to use this box to spoof packets leaving the firewall itself, and hence we only want to allow traffic from the IP addresses assigned to the firewall itself. We would most likely implement this by adding rules that ACCEPT all packets leaving the firewall in case they come from one of the IP addresses assigned to the firewall, and if not they will be dropped by the default policy in the OUTPUT chain.

8.6.4 Setting up default policies

Quite early on in the process of creating our ruleset, we set up the default policies. We set up the default policies on the different chains with a fairly simple command, as described below.

iptables [-P {chain} {policy}]

The default policy is used every time the packets do not match a rule in the chain. For example, lets say we get a packet that match no single rule in our whole ruleset. If this happens, we must decide what should happen to the packet in question, and this is where the default policy comes into the picture. The default policy is used on all packets that does not match with any other rule in our ruleset.

8.6.5 Setting up user specified chains in the filter table

It is now time that we take care of setting up all the rules and chains that we wish to create and to use, as well as all of the rulesets within the chains.

After this, we create the different special chains that we want to use with the **-N** command. The new chains are created and set up with no rules inside of them. The chains we will use are, as previously described, icmp_packets, tcp_packets, udpincoming_packets and the allowed chain, which is used by the tcp_packets chain. Incoming packets on **$INET_IFACE**, of ICMP type, will be redirected to the chain

icmp_packets. Packets of TCP type, will be redirected to the tcp_packets chain and incoming packets of UDP type from **$INET_IFACE** go to udpincoming_packets chain.

8.6.5.1 The bad_tcp_packets chain

The bad_tcp_packets chain is devoted to contain rules that inspects incoming packets for malformed headers or other problems. As it is, we have only chosen to include a packet filter which blocks all incoming TCP packets that are considered as **NEW** but does not have the SYN bit sent.

8.6.5.2 The allowed chain

If a packet comes in on **$INET_IFACE** and is of TCP type, it travels through the tcp_packets chain and if the connection is against a port that we want to allow traffic on, we want to do some final checks on it to see if we actually do want to allow it or not. All of these final checks are done within the allowed chain.

First of all, we check if the packet is a SYN packet. If it is a SYN packet, it is most likely to be the first packet in a new connection so, of course, we allow this. Then we check if the packet comes from an **ESTABLISHED** or **RELATED** connection, if it does, then we, again of course, allow it. An **ESTABLISHED** connection is a connection that has seen traffic in both directions, and since we have seen a SYN packet, the connection then must be in state **ESTABLISHED**, according to the state machine. The last rule in this chain will **DROP** everything else. In this case this pretty much means everything that has not seen traffic in both directions, ie, we didn't reply to the SYN packet, or they are trying to start the connection with a non SYN packet. There is *no* practical use of not starting a connection with a SYN packet, except to portscan people pretty much. There is no currently available TCP/IP implementation that supports opening a TCP connection with something else than a SYN packet ,hence **DROP** it since it is 99% sure to be a portscan.

8.6.5.3 The TCP chain

The tcp_packets chain specifies what ports that are allowed to use on the firewall from the Internet. A **tcp_packets** tells **iptables** in which chain to add the new rule, the rule will be added to the end of the chain. **-p TCP** tells it to match TCP packets and **-s 0/0** matches all source addresses from 0.0.0.0 with netmask 0.0.0.0, in other words *all* source addresses. **--dport 21** means destination port 21, in other words if the packet is destined for port 21 they also match. If all the criteria are matched, then the packet will be targeted for the allowed chain. If it doesn't match any of the rules, they will be passed back to the original chain that sent the packet to the tcp_packets chain.

As it is now, I allow TCP port 21, or FTP control port, which is used to control FTP. If we do not want to allow FTP at all, we can unload the ip_conntrack_ftp module and delete the **$IPTABLES -A tcp_packets -p TCP -s 0/0 --dport 21 -j allowed** line from the script.

Port 22 is SSH, which is much better than allowing telnet on port 23. It is always a bad idea to give others than yourself any kind of access to a firewall box. Firewalls should always be kept to a bare minimum and no more.

Port 80 is HTTP, in other words our web server, we can delete it if we do not want to run a web server directly on the firewall.

8.6.5.4 The UDP chain

If we do get a UDP packet on the INPUT chain, we send them on to udpincoming_packets where we once again do a match for the UDP protocol with **-p UDP** and then match everything with a source address of 0.0.0.0 and netmask 0.0.0.0, in other words everything again. Except this, we only accept specific UDP ports that we want to be open for hosts on the Internet. We only need to open up ports on our host if we are to run a server on any UDP port, such as DNS etc. Packets that are entering the firewall and that are part of an already established connection (by our local network) will automatically be accepted back in by the --state ESTABLISHED,RELATED rules at the top of the INPUT chain.

8.6.5.5 The ICMP chain

This is where we decide what ICMP types to allow. If a packet of ICMP type comes in on eth0 on the INPUT chain, we then redirect it to the `icmp_packets` chain as explained before. Here we check what kind of ICMP types to allow. For now, I only allow incoming ICMP Echo requests, TTL equals 0 during transit and TTL equals 0 during reassembly. The reason that we do not allow any other ICMP types per default here, is that almost all other ICMP types should be covered by the RELATED state rules.

The reason that I allow these ICMP packets are as follows, Echo Requests are used to request an echo reply, which in turn is used to mainly ping other hosts to see if they are available on any of the networks. Without this rule, other hosts will not be able to ping us to see if we are available on any network connection.

8.6.5.6 INPUT chain

The INPUT chain uses mostly other chains to do the hard work. This way we do not get too much load from iptables, and it will work much better. This is done by checking for specific details that should be the same for a lot of different packets, and then sending those packets into specific user specified chains. By doing this, we can split down our ruleset to contain much less rules that needs to be traversed by each packet and hence the firewall will be put through a lot less overhead by packet filtering.

First of all we do certain checks for bad packets. This is done by sending all TCP packets to the bad_tcp_packets chain. This chain contains a few rules that will check for badly formed packets or other anomalies that we do not want to accept.

At this point we start looking for traffic from generally trusted networks. These include the local network adapter and all traffic coming from there, all traffic to and from our loopback interface, including all our currently assigned IP addresses (this means all of them, including our Internet IP address). As it is, we have chosen to put the rule that allows LAN activity to the firewall at the top, since our local network generates more traffic than the Internet connection. This allows for less overhead used to try and match each packet with each rule and it is always a good idea to look through what kind of traffic mostly traverses the firewall. By doing this, we can shuffle around the rules to be more efficient, leading to less overhead on the firewall and less congestion on our network.

After this, We match all TCP packets in the INPUT chain that comes in on the **SINET_IFACE** interface, and send those to the `tcp_packets`, which was previously described. Now we do the same match for UDP packets on the **SINET_IFACE** and send those to the udpincoming_packets chain, and after this all ICMP packets are sent to the icmp_packets chain. Normally, a firewall would be hardest hit by TCP packets, then UDP and last of them all ICMP packets.

Before we hit the default policy of the INPUT chain, we log it so we may be able to find out about possible problems and/or bugs. Though, we do not log more than 3 packets per minute as we do not want to flood our logs, also we set a prefix to all log entries so we know where it came from.

Everything that has not yet been caught will be **DROP**'ed by the default policy on the INPUT chain.

8.6.5.7 FORWARD chain

The FORWARD chain contains quite few rules here. We have a single rule which sends all packets to the bad_tcp_packets chain, which was also used in the INPUT chain as described previously.

After this first check for bad TCP packets, we have the main rules in the FORWARD chain. The first rule will allow all traffic from our **SLAN_IFACE** to any other interface to flow freely, without restrictions. This rule will in other words allow all traffic from our LAN to the Internet. The second rule will allow **ESTABLISHED** and **RELATED**traffic back through the firewall. This will in other words allow packets belonging to connections that was initiated from our internal network to flow freely back to our local network. These rules are required for our local network to be able to access the Internet, since the default policy of the FORWARD chain was previously set to **DROP**. I will allow hosts on our local network to connect to hosts on the internet, but at the same time block hosts on the internet to connect to the hosts on our internal network.

Finally we also have a logging rule which will log packets that are not allowed in one or another way to pass through the FORWARD chain. This is exactly the same rule as the one used in the INPUT chain except for the logging prefix, **"IPT FORWARD packet**

died: ". The logging prefix is mainly used to separate log entries, and may be used to distinguish log entries to find out where the packet was logged from and some header options.

8.6.5.8 OUTPUT chain

Since the system is used as a Firewall and a workstation currently, I allow almost everything that goes out from it that has a source address **$LOCALHOST_IP**, **$LAN_IP** or **$STATIC_IP**. Last of all we log everything that gets dropped. Finally we **DROP** the packet in the default policy.

8.6.5.9 PREROUTING chain of the nat table

The PREROUTING chain is pretty much what it says. it does network adress translation on packets before they actually hit the routing decision that sends them onwards to the INPUT or FORWARD chains in the filter table.

8.6.5.10 Starting SNAT and the POSTROUTING chain

First of all we add a rule to the nat table, in the POSTROUTING chain that will NAT all packets going out on our interface connected to the Internet. This would be eth0. So all packets that match this rule will be Source Network Address Translationed to look as it came from firewall's interface.

8.7 Script

```
#!/bin/bash
#
## Internet Configuration.
#
INET_IP="192.168.0.214"
INET_IFACE="eth0"

## Local Area Network configuration.
#
# LAN's IP range and localhost IP. /24 means to only use the first 24
# bits of the 32 bit IP adress. the same as netmask 255.255.255.0
#
LAN_IP="192.168.201.4"
LAN_IP_RANGE="192.168.201.0/24"
LAN_BCAST_ADRESS="192.168.201.255"
LAN_IFACE="eth1"

## Reserved/Private IP Addresses ##
RESERVED_NET="0.0.0.0/8 1.0.0.0/8 2.0.0.0/8 5.0.0.0/8 7.0.0.0/8 23.0.0.0/8 \
              27.0.0.0/8 31.0.0.0/8 36.0.0.0/8 37.0.0.0/8 39.0.0.0/8 \
              41.0.0.0/8 42.0.0.0/8 58.0.0.0/8 59.0.0.0/8 60.0.0.0/8 \
              67.0.0.0/8 68.0.0.0/8 69.0.0.0/8 70.0.0.0/8 71.0.0.0/8 72.0.0.0/8
              73.0.0.0/8 74.0.0.0/8 75.0.0.0/8 76.0.0.0/8 77.0.0.0/8 78.0.0.0/8 \
              79.0.0.0/8 80.0.0.0/8 81.0.0.0/8 82.0.0.0/8 83.0.0.0/8 \
              84.0.0.0/8 85.0.0.0/8 86.0.0.0/8 87.0.0.0/8 88.0.0.0/8 89.0.0.0/8 \
              90.0.0.0/8 91.0.0.0/8 92.0.0.0/8 93.0.0.0/8 94.0.0.0/8 \
             95.0.0.0/8 96.0.0.0/8 97.0.0.0/8 98.0.0.0/8 99.0.0.0/8 100.0.0.0/8
             101.0.0.0/8 102.0.0.0/8 103.0.0.0/8 104.0.0.0/8 105.0.0.0/8
             106.0.0.0/8 107.0.0.0/8 108.0.0.0/8 109.0.0.0/8 110.0.0.0/8 \
             111.0.0.0/8 112.0.0.0/8 113.0.0.0/8 114.0.0.0/8 115.0.0.0/8 \
              116.0.0.0/8 117.0.0.0/8 118.0.0.0/8 119.0.0.0/8 120.0.0.0/8 \
              121.0.0.0/8 122.0.0.0/8 123.0.0.0/8 124.0.0.0/8 125.0.0.0/8 \
              126.0.0.0/8 127.0.0.0/8 197.0.0.0/8 201.0.0.0/8 219.0.0.0/8 \
              220.0.0.0/8 221.0.0.0/8 222.0.0.0/8 223.0.0.0/8 240.0.0.0/8\
              241.0.0.0/8 242.0.0.0/8 243.0.0.0/8 244.0.0.0/8 245.0.0.0/8 \
              246.0.0.0/8 247.0.0.0/8 248.0.0.0/8 249.0.0.0/8 250.0.0.0/8 \
              251.0.0.0/8 252.0.0.0/8 253.0.0.0/8 254.0.0.0/8 255.0.0.0/8"
```

```
## Localhost Configuration
#
LO_IFACE="lo"
LO_IP="127.0.0.1"
##IPTables Configuration.
#
IPTABLES="/sbin/iptables"


# #Reset the default policies in the filter table.
#
$IPTABLES -P INPUT ACCEPT
$IPTABLES -P FORWARD ACCEPT
$IPTABLES -P OUTPUT ACCEPT

##Reset the default policies in the nat table.
#
$IPTABLES -t nat -P PREROUTING ACCEPT
$IPTABLES -t nat -P POSTROUTING ACCEPT
$IPTABLES -t nat -P OUTPUT ACCEPT

##Reset the default policies in the mangle table.
#
$IPTABLES -t mangle -P PREROUTING ACCEPT
$IPTABLES -t mangle -P OUTPUT ACCEPT

## Flush all the rules in the filter and nat tables.
#
$IPTABLES -F
$IPTABLES -t nat -F
$IPTABLES -t mangle -F

##Erase all chains that's not default in filter and nat table.
#
$IPTABLES -X
$IPTABLES -t nat -X
$IPTABLES -t mangle -X


# #Other Configuration##.
#
## /proc Configuration
#
echo "1" > /proc/sys/net/ipv4/icmp_ignore_bogus_error_responses
#
```

```
                !
##Set local port range
echo "32768 61000" > /proc/sys/net/ipv4/ip_local_port_range
#
##Reducing DoS by inducing timeouts
echo 30 > /proc/sys/net/ipv4/tcp_fin_timeout
echo 1800 > /proc/sys/net/ipv4/tcp_keepalive_time
echo 1 > /proc/sys/net/ipv4/tcp_window_scaling
echo 0 > /proc/sys/net/ipv4/tcp_sack
echo 1280 > /proc/sys/net/ipv4/tcp_fin_timeout
echo 30 > /proc/sys/net/ipv4/tcp_max_syn_backlog

## Set the maximum number of connections to track. (Kernel Default: 2048)
if [ -e /proc/sys/net/ipv4/ip_conntrack_max ]; then
        echo "4096" > /proc/sys/net/ipv4/ip_conntrack_max
fi
                !

## Local port range for TCP/UDP connections
if [ -e /proc/sys/net/ipv4/ip_local_port_range ]; then
        echo -e "32768\t61000" > /proc/sys/net/ipv4/ip_local_port_range
fi

## Disable TCP Explicit Congestion Notification Support
#       if [ -e /proc/sys/net/ipv4/tcp_ecn ]; then
#               echo "0" > /proc/sys/net/ipv4/tcp_ecn
#       fi

## Disable source routing of packets
if [ -e /proc/sys/net/ipv4/conf/all/accept_source_route ]; then
        for i in /proc/sys/net/ipv4/conf/*/accept_source_route; do
                echo "0" > $i;
done            !
fi

## Enable rp_filter
if [ -e /proc/sys/net/ipv4/conf/all/rp_filter ]; then
        for i in /proc/sys/net/ipv4/conf/*/rp_filter; do
                echo "1" > $i;
        done
fi

## Kill Timestamps
if [ -e /proc/sys/net/ipv4/tcp_timestamps ]; then
        echo "0" > /proc/sys/net/ipv4/tcp_timestamps
fi


                !
```

```
## Ignore any broadcast icmp echo requests
if [ -e /proc/sys/net/ipv4/icmp_echo_ignore_broadcasts ]; then
        echo "1" > /proc/sys/net/ipv4/icmp_echo_ignore_broadcasts
fi              !

## Ignore all icmp echo requests on all interfaces
#       if [ -e /proc/sys/net/ipv4/icmp_echo_ignore_all ]; then
#               echo "0" > /proc/sys/net/ipv4/icmp_echo_ignore_all
#       fi

## Log packets with impossible addresses to kernel log.
if [ -e /proc/sys/net/ipv4/conf/all/log_martians ]; then
        echo "1" > /proc/sys/net/ipv4/conf/all/log_martians
fi

## Don't accept ICMP redirects
## Disable on all interfaces
#       if [ -e /proc/sys/net/ipv4/conf/all/accept_redirects ]; then
#               echo "0" > /proc/sys/net/ipv4/conf/all/accept_redirects
#       fi      !

## Disable only on the external interface.
if [ -e /proc/sys/net/ipv4/conf/$INET_IP/accept_redirects ]; then
        echo "0" > /proc/sys/net/ipv4/conf/$INET_IP/accept_redirects
fi

##For dynamic addresses
#       if [ -e /proc/sys/net/ipv4/ip_dynaddr ]; then
#               echo "1" > /proc/sys/net/ipv4/ip_dynaddr
#       fi

##Enable syncookie protection
if [ -r /proc/sys/net/ipv4/tcp_syncookies ]; then
        echo "Enabling tcp_syncookies"
        echo "1" > /proc/sys/net/ipv4/tcp_syncookies
fi              !

## Enable IP Forwarding
if [ -e /proc/sys/net/ipv4/ip_forward ]: then
        echo "Enabling ipforwarding"
        echo "1" > /proc/sys/net/ipv4/ip_forward
else
        echo "ERROR: /proc/sys/net/ipv4/ip_forward does not exist"
        echo "(check it)"
        echo
fi
```



```
                !
```

```
###########
#
# Rules set up.
#
#
### Filter table
#
##Set policies
#
$IPTABLES -P INPUT DROP
$IPTABLES -P OUTPUT DROP
$IPTABLES -P FORWARD DROP

## Create userspecified chains & Create chain for bad tcp packets
## Create separate chains for ICMP, TCP and UDP to traverse
#
$IPTABLES -N allowed
$IPTABLES -N icmp_packets
$IPTABLES -N tcp_packets
$IPTABLES -N udpincoming_packets
$IPTABLES -N bad_tcp_packets

############################
## Special Chain SRC_EGRESS
#
## Rules to Provide Egress Filtering Based on Source IP Address.
#
$IPTABLES -N SRC_EGRESS
$IPTABLES -F SRC_EGRESS
#
## Class A Reserved
$IPTABLES -A SRC_EGRESS -s 10.0.0.0/8 -j DROP
#
## Class B Reserved
$IPTABLES -A SRC_EGRESS -s 172.16.0.0/12 -j DROP
#
## Class C Reserved
$IPTABLES -A SRC_EGRESS -s 192.168.0.0/16 -j DROP
#
## Class D Reserved
$IPTABLES -A SRC_EGRESS -s 224.0.0.0/4 -j DROP
#
```

```
!

## Class E Reserved
$IPTABLES -A SRC_EGRESS -s 240.0.0.0/5 -j DROP
#
for NET in $RESERVED_NET; do
$IPTABLES -A SRC_EGRESS -s $NET -j DROP
done
##------------------------------------------------------------------------##


############################
## Special Chain DST_EGRESS
## Rules to Provide Egress Filtering Based on Destination IP Address.
#
$IPTABLES -N DST_EGRESS
$IPTABLES -F DST_EGRESS
#
## Class A Reserved
$IPTABLES -A DST_EGRESS -d 10.0.0.0/8 -j DROP
#
## Class B Reserved
$IPTABLES -A DST_EGRESS -d 172.16.0.0/12 -j DROP
#
## Class C Reserved
$IPTABLES -A DST_EGRESS -d 192.168.0.0/16 -j DROP
#
## Class D Reserved
$IPTABLES -A DST_EGRESS -d 224.0.0.0/4 -j DROP
#
## Class E Reserved
$IPTABLES -A DST_EGRESS -d 240.0.0.0/5 -j DROP
#
for NET in $RESERVED_NET; do
$IPTABLES -A DST_EGRESS -d $NET -j DROP
done
##------------------------------------------------------------------------##

## Create content in userspecified chains
#
# #bad_tcp_packets chain
#
$IPTABLES -A bad_tcp_packets -p tcp ! --syn -m state --state NEW -j LOG\
--log-prefix "New not syn:"
$IPTABLES -A bad_tcp_packets -p tcp ! --syn -m state --state NEW -j DROP
#

!
```

```
# #allowed chain
#
$IPTABLES -A allowed -p TCP --syn -j ACCEPT
$IPTABLES -A allowed -p TCP -m state --state ESTABLISHED,RELATED -j ACCEPT
$IPTABLES -A allowed -p TCP -j DROP
#

## TCP rules
#
$IPTABLES -A tcp_packets -p TCP -s 0/0 --dport 21 -j allowed
$IPTABLES -A tcp_packets -p TCP -s 0/0 --dport 22 -j allowed
$IPTABLES -A tcp_packets -p TCP -s 0/0 --dport 80 -j allowed
#

## UDP ports
#
#$IPTABLES -A udpincoming_packets -p UDP -s 0/0 --destination-port 53 -j ACCEPT
#$IPTABLES -A udpincoming_packets -p UDP -s 0/0 --destination-port 123 -j ACCEPT
$IPTABLES -A udpincoming_packets -p UDP -s 0/0 --destination-port 2074 -j ACCEPT

## ICMP rules
#

## Echo Reply (pong)
$IPTABLES -A icmp_packets -p icmp --icmp-type echo-reply -j ACCEPT

## Destination Unreachable
$IPTABLES -A icmp_packets -p icmp --icmp-type destination-unreachable \
        -j ACCEPT

## Accept Pings##

$IPTABLES -A icmp_packets -p icmp --icmp-type echo-request -j ACCEPT

## Accept Pings at the rate of one per second ##

#$IPTABLES -A icmp_packets -p icmp --icmp-type echo-request \

#-m limit --limit 1/second -j ACCEPT

## LOG all pings ##

#$IPTABLES -A icmp_request -p icmp --icmp-type echo-request \

#-m limit --limit 5/minute -j LOG --log-level $LOG_LEVEL \
```

```
#--log-prefix "PING:"

## TTL Exceeded (traceroute)

$IPTABLES -A icmp_request -p icmp --icmp-type time-exceeded -j ACCEPT

$IPTABLES -A icmp_packets -p ICMP -s 0/0 --icmp-type 8 -j ACCEPT
$IPTABLES -A icmp_packets -p ICMP -s 0/0 --icmp-type 11 -j ACCEPT


###
# INPUT chain
#
# DROP packets not destined for the internal IP address of the firewall.

$IPTABLES -A INPUT -i $LAN_IFACE -d ! $LAN_IP -j DROP

## Filter out Reserved/Private IP addresses based on source IP.

$IPTABLES -A INPUT -i $INET_IFACE -j SRC_EGRESS

## Filter out Reserved/Private IP addresses based on destination IP.

$IPTABLES -A INPUT -i $INET_IFACE -j DST_EGRESS

$IPTABLES -A INPUT -p tcp -j bad_tcp_packets

#
# Rules for special networks not part of the Internet
#
$IPTABLES -A INPUT -p ALL -i $LAN_IFACE -s $LAN_IP_RANGE -j ACCEPT
$IPTABLES -A INPUT -p ALL -i $LO_IFACE -s $LO_IP -j ACCEPT
$IPTABLES -A INPUT -p ALL -i $LO_IFACE -s $LAN_IP -j ACCEPT
$IPTABLES -A INPUT -p ALL -i $LO_IFACE -s $INET_IP -j ACCEPT
$IPTABLES -A INPUT -p ALL -i $LAN_IFACE -d $LAN_BCAST_ADRESS -
j ACCEPT

#
# Rules for incoming packets from the internet.
#

$IPTABLES  -A INPUT -p ALL –d $INET_IP –m state  --state ESTABLISHED,\

RELATED  -j ACCEPT
```

```
$IPTABLES -A INPUT -p TCP -i $INET_IFACE -j tcp_packets
$IPTABLES -A INPUT -p UDP -i $INET_IFACE -j udpincoming_packets
$IPTABLES -A INPUT -p ICMP -i $INET_IFACE -j icmp_packets

#
# Log packets that don't match the above.
#

$IPTABLES -A INPUT -m limit --limit 3/minute --limit-burst 3 -j LOG \
--log-level DEBUG --log-prefix "IPT INPUT packet died: "

#
#  FORWARD chain
#
## DROP any attempted NEW connections to the internal network.

$IPTABLES -A FORWARD -i $INET_IFACE -d $LAN_IP_RANGE -m state \

--state NEW -j DROP

## DROP any outbound traffic to the internal network that is trying to

 ## establish a NEW connection.

$IPTABLES -A FORWARD -o $LAN_IFACE -d $LAN_IP_RANGE \

-m state --state NEW -j DROP

## DROP echo reply packets coming into the internal interface.

$IPTABLES -A FORWARD -o $LAN_IFACE -p icmp --icmp-type echo-request \

-j DROP

## DROP anything not headed for the internal network.

$IPTABLES -A FORWARD -i $INET_IFACE -d ! $LAN_IP_RANGE -j DROP

## Filter out Reserved/Private IP addresses based on Source IP.

$IPTABLES -A FORWARD -i $INET_IFACE -j SRC_EGRESS

$IPTABLES -A FORWARD -o $INET_IFACE -s ! $LAN_IP_RANGE –j \
SRC_EGRESS
```

```
## Filter out Reserved/Private IP addresses based on destination IP.

$IPTABLES -A FORWARD -o $INET_IFACE -j DST_EGRESS

## Filter out Reserved/Private IP addresses based on Destination IP.

$IPTABLES -A FORWARD -i $LAN_IFACE -j DST_EGRESS

$IPTABLES -A FORWARD -o $LAN_IFACE -j SRC_EGRESS



# Bad TCP packets we don't want
#

$IPTABLES -A FORWARD -p tcp -j bad_tcp_packets

#
# Accept the packets we actually want to forward
#

$IPTABLES -A FORWARD -i $LAN_IFACE -j ACCEPT
$IPTABLES -A FORWARD -m state --state ESTABLISHED,RELATED -j ACCEPT

#
# Log packets that don't match the above.
#

$IPTABLES -A FORWARD -m limit --limit 3/minute --limit-burst 3 -j LOG \
--log-level DEBUG --log-prefix "IPT FORWARD packet died: "

#
# OUTPUT chain
#
## DROP anything not coming from the firewall.

$IPTABLES -A OUTPUT -o $LAN_IFACE -s ! $LAN_IP -j DROP

## OUTPUT on the external interface

## Filter out Reserved/Private IP addresses based on source IP.

$IPTABLES -A OUTPUT -o $EXTERNAL -j SRC_EGRESS
```

```
## Filter out Reserved/Private IP addresses based on destination IP.

$IPTABLES -A OUTPUT -o $EXTERNAL -j DST_EGRESS

# Bad TCP packets we don't want.
#
                !

$IPTABLES -A OUTPUT -p tcp -j bad_tcp_packets

#
# Special OUTPUT rules to decide which IP's to allow.
#

$IPTABLES -A OUTPUT -p ALL -s $LO_IP -j ACCEPT
$IPTABLES -A OUTPUT -p ALL -s $LAN_IP -j ACCEPT
$IPTABLES -A OUTPUT -p ALL -s $INET_IP -j ACCEPT

#
# Log packets that don't match the above.
#

$IPTABLES -A OUTPUT -m limit --limit 3/minute --limit-burst 3 -j LOG \
--log-level DEBUG --log-prefix "IPT OUTPUT packet died: "

######
#NAT table
#
# Set policies &
#Create user specified chains &  Create content in user specified chains
#
# PREROUTING chain
#POSTROUTING chain
# Enable simple IP Forwarding and Network Address Translation
#
$IPTABLES -t nat -A POSTROUTING -o $INET_IFACE -j SNAT --to-
source $INET_IP
                !

##  END
```

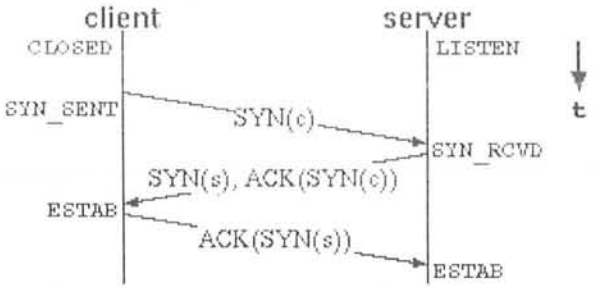

!

Glossary

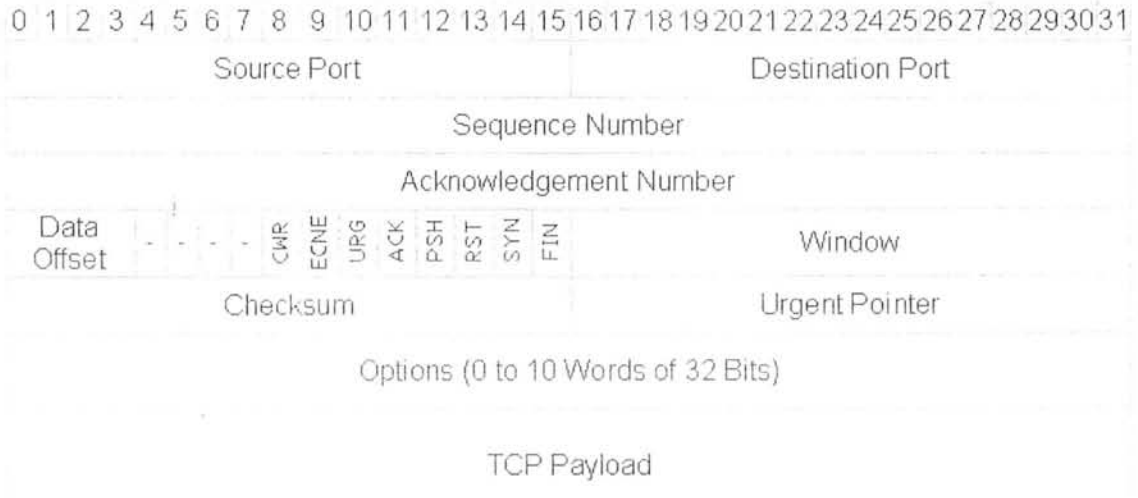
access control - The prevention of unauthorized use of a resource including the prevention of use of a resource in an unauthorized manner
bastion host - A heavily secured host.
denial of service - The unauthorized prevention of authorized access to resources or the delaying of time-critical operations
DNS spoofing - Assuming the DNS name of another system by either corrupting the name service cache of a victim system, or by compromising a domain name server for a valid domain.
host - Any computer that is connected to a network.
ICMP - Internet Control Message Protocol (RFC 792)
Insider Attack - An attack originating from inside a protected network.
IP - Internet Protocol (RFC 791)
IP body - contains the actual data in an IP packet
IP packet - an IP packet is made up of an IP header and an IP body
IP header - an IP header contains meta-data about the IP body (diagram from RFC 791)

IP Spoofing - An attack whereby a system attempts to illicitly impersonate another system by using its IP network address.
IPC - Interprocess communication
Logging - The process of storing information about events that occurred on the firewall or network.
Network Address Translation (NAT) - A process that modifies either the source IP address or destination IP address of an IP packet.
packet - A unit of data exchange between hosts.
RPC - Remote Procedure Call.

server - a host that will provide a network service to other hosts (i.e. accept new connections)
TCP - Transmission Control Protocol (RFC 793)
TCP handshake - a three step process that every (successful) TCP connection must follow



TCP header - a TCP header contains meta-data about the TCP body (diagram from RFC 793)



TCP/IP - even though commonly spoken as if it's a single protocol, TCP/IP is actually a TCP tunnel over the IP layer
Virtual Private Network - A network that appears to be a single protected network behind firewalls, which actually encompasses encrypted virtual links over untrusted networks.

Bibliography

1. Douglas E. Comer. **Internetworking with TCP/IP: Principles, Protocols, and Architecture**. Prentice-Hall, Englewood Cliffs, NJ, 1991.

2. Steven M. Bellovin. **Security Problems in the TCP/IP ProtocolSuite**. Computer Communications Review, April 1989.

3. Frederick Avolio and Marcus Ranum. **A Network Perimeter With Secure Internet Access**. In Internet Society Symposium on Network and Distributed System Security, February 2-4 1994.

4. Computer Emergency Response Team/Coordination Center. **Ongoing Network Monitoring Attacks**. Available from FIRST.ORG, file pub/alerts/cert9401 .txt.

5. NIST. Security in Open Systems. Special Publication. National Institute of Standards and Technology, September 1994.

6. Marcus Ranum. **Thinking About Firewalls**. In SANS-II Conference, April 1993.

7. X/Open. **Security Survival**: A Source Book from The Open Group X/Open Company Ltd., U.K. April 1996.

8. **The Netfilter Project** Homepage (http://netfilter.samba.org/) Netfilter Mailing List (http://lists.samba.org/pipermail/netfilter/)