Designing and implementing a firewall in a campus network

ABHAY MALHOTRA (071328)

HITESH KHARBANDA (071350)

NEHA BHATIA (071501)

NIKHIL SINGHAL (071326)

UNDER THE SUPERVISION OF

BRIG(RETD). S.P. GHRERA

Submitted in partial fulfillment of the Degree of

Bachelor of Technology

DEPARTMENT OF CSE & IT

JAYPEE UNIVERSITY OF INFORMATION TECHNOLOGY,

WAKNAGHAT

TABLE OF CONTENTS

CERTIFICATE

This is to certify that the work entitled **"IMPLEMENTING A FIREWALL"** submitted by **"Abhay Malhotra, Hitesh Kharbanda, Neha Bhatia and Nikhil Singhal"** in partial fulfillment for the award of degree of B.Tech of Jaypee University of Information Technology, Waknaghat has been carried out under my supervision. This work has not been submitted partially or wholly to any other University or Institute for the award of this or any other degree or diploma.

Signature of Supervisor :

Name of Supervisor : Brig(Retd). S.P. Ghrera

Designation : HOD (CSE Dept.)

Date : 21st May' 2011

ACKNOWLEDGEMENT

No one can achieve success without acknowledging the help of others. The wise and confident, acknowledge any help with gratitude.

It is almost inevitable to incur indebtedness to all who generously helped by sharing their invaluable time and rich experience with us, without which this project would have never been on its way to successful completion.

We would like to express our sincere thanks to Brig(Retd). S.P.Ghrera and Dr. Satish Chandra who played a great role of a mentor in assisting our work and providing a constant encouragement during this period. He has been immensely contribute with his ideas, constructive criticism and motivation which were the guiding light during the entire tenure of this work. Also, he has been benevolent enough to lend us help and spare his valuable time whenever needed.

Name: Abhay Malhotra
Roll No: 071328

Name: Hitesh Kharbanda
Roll No: 071350

Name: Neha Bhatia
Roll No: 071501

Name: Nikhil Singhal
Roll No: 071326

SUMMARY

The project implements a full-featured network security solution in the form of a four types of completely self-sufficient firewall system. The firewall and proxying services operate over a modularized Linux OS architecture. In addition, this application infers the occurrence of abnormal activity occurring on the network by processing the firewall audit records, given a set of rule signatures, and generates quantitative alerts regarding the anomalous behaviour. It thus provides the network administrator with an advanced functionality for monitoring and tracking large-volume networks. The rules implemented can prevent many situations and anti spoofing rules and vpn rules are also added to make it better.

LIST OF FIGURES

LIST OF ABBREVIATIONS AND SYMBOLS

Tcp	Transmission Control Protocol
OSI	Open System Interface
Icmp	Internet Control Message Protocol
Http	Hypertext Transfer Protocol
SSH	Secure Shell
FTP	File Transfer Protocol
IP	Internet Protocol
Src	source
dport	Destination port
NAT	Network Address Translation

CHAPTER 1
INTRODUCTION

1.1 NETWORK SECURITY:

With the widespread use of distributed and networked computer systems, protecting data being stored on such systems and being transmitted between them has become necessary. Network security refers to this important aspect of computer security that deals with securing network communication and storage. With the spread of the Internet, the need for dependable network security solutions has become increasingly evident. Network security applications thus need to perform both preventive as well as remedial functions in today's networks. Some of these key functions of network security services are listed below:-

- **Confidentiality** – This function ensures that information transactions are accessible only to authorized parties.
- **Authentication** – Ensures that the origin of a message or electronic document is correctly identified, with the assurance that the identity is not false.
- **Integrity** – Ensures that only the authorized parties are able to modify system assets and transmitted information.
- **Non-repudiation** – This function requires that neither the sender nor receiver of a message can deny its transmission.
- **Access Control** – Ensures that access to secure information is controlled in a graded manner.
- **Availability** – This function requires that the computer system assets be available to authorized parties as and when requested.

1.2 FIREWALL:

1.2.1 What is a firewall?

A firewall protects networked computers from intentional hostile intrusion that could compromise confidentiality or result in data corruption or denial of service. It may be a hardware device or a software program running on a secure host computer. In either case, it must have at least two network interfaces, one for the network it is intended to protect, and one for the network it is exposed to. A firewall sits at the junction point or gateway between the two networks, usually a private network and a public network such as the Internet. The earliest firewalls were simply routers. The term firewall comes from the fact that by segmenting a network into different physical sub networks, they limited the damage that could spread from one subnet to another just like fire doors or firewalls.

1.2.2 What does a firewall do?

A firewall examines all traffic routed between the two networks to see if it meets certain criteria. If it does, it is routed between the networks, otherwise it is stopped. A firewall filters both inbound and outbound traffic. It can also manage public access to private networked resources such as host applications. It can be used to log all attempts to enter the private network and trigger alarms when hostile or unauthorized entry is attempted. Firewalls can filter packets based on their source and destination addresses and port numbers. This is known as address filtering. Firewalls can also filter specific types of network traffic. This is also known as protocol filtering because the decision to forward or reject traffic is dependent upon the protocol used, for example HTTP, ftp or telnet. Firewalls can also filter traffic by packet attribute or state.

1.2.3 What can't a firewall do?

A firewall cannot prevent individual users with modems from dialling into or out of the network, bypassing the firewall altogether. Employee misconduct or carelessness cannot be controlled by firewalls. Policies involving the use and misuse of passwords and user accounts must be strictly enforced. These are management issues that should be raised during the planning of any security policy but that cannot be solved with firewalls alone. The arrest of the Phone master's cracker ring brought these security issues to light. Although they were accused of breaking into information systems run by AT&T Corp., British Telecommunications Inc., GTE Corp., MCI WorldCom, South western Bell, and Sprint Corp, the group did not use any high tech methods such as IP spoofing (see question. They used a combination of social engineering and dumpster diving. Social engineering involves skills not unlike those of a confidence trickster. People are tricked into revealing sensitive information. Dumpster diving or garb logy, as the name suggests, is just plain old looking through company trash. Firewalls cannot be effective against either of these techniques.

1.2.4 Who needs a firewall?

Anyone who is responsible for a private network that is connected to a public network needs firewall protection. Furthermore, anyone who connects so much as a single computer to the Internet via modem should have personal firewall software. Many dial-up Internet users believe that anonymity will protect them. They feel that no malicious intruder would be motivated to break into their computer. Dial up users who have been victims of malicious attacks and who have lost entire days of work, perhaps having to reinstall their operating system, know that this is not true. Irresponsible pranksters can use automated robots to scan random IP addresses and attack whenever the opportunity presents itself.

1.2.5 How does a firewall work?

There are two access denial methodologies used by firewalls. A firewall may allow all traffic through unless it meets certain criteria, or it may deny all traffic unless it meets certain criteria. The type of criteria used to determine whether traffic should be allowed through varies from one type of firewall to another. Firewalls may be concerned with the type of traffic, or with source or destination addresses and ports.

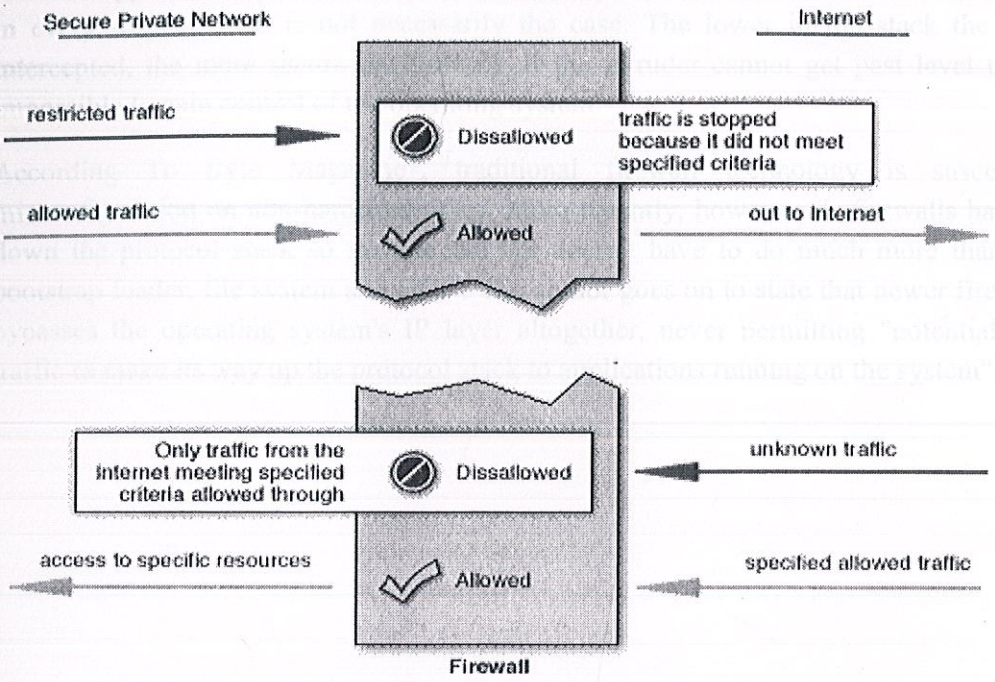

Figure 1. Firewall

They may also use complex rule bases that analyse the application data to determine if the traffic should be allowed through. How a firewall determines what traffic to let through depends on which network layer it operates at. A discussion on network layers and architecture follows.

1.2.6 What are the OSI and TCP/IP Network models?

To understand how firewalls work it helps to understand how the different layers of a network interact. Network architecture is designed around a seven layer model. Each layer has its own set of responsibilities, and handles them in a well-defined manner. This enables networks to mix and match network protocols and physical supports. In a given network, a single protocol can travel over more than one physical support (layer one) because the physical layer has been dissociated from the protocol layers (layers three to seven). Similarly, a single physical cable can carry more than one protocol. The TCP/IP model is older than the OSI industry standard model which is why it does not comply in every respect. The first four layers are so closely analogous to OSI layers however that interoperability is a day to day reality. Firewalls operate at different layers to use different criteria to restrict traffic. The lowest layer at which a firewall can work is layer three. In the OSI model this is the network layer. In TCP/IP it is the Internet Protocol layer. This layer is concerned with routing packets to their destination. At this layer a firewall can determine whether a packet is from a trusted source, but cannot be concerned with what it contains or

what other packets it is associated with. Firewalls that operate at the transport layer know a little more about a packet, and are able to grant or deny access depending on more sophisticated criteria. At the application level, firewalls know a great deal about what is going on and can be very selective in granting access.

It would appear then, that firewalls functioning at a higher level in the stack must be superior in every respect. This is not necessarily the case. The lower in the stack the packet is intercepted, the more secure the firewall. If the intruder cannot get past level three, it is impossible to gain control of the operating system.

According To Byte Magazine*, traditional firewall technology is susceptible to misconfiguration on non-hardened OSes. More recently, however, "...firewalls have moved down the protocol stack so far that the OS doesn't have to do much more than act as a bootstrap loader, file system and GUI". The author goes on to state that newer firewall code bypasses the operating system's IP layer altogether, never permitting "potentially hostile traffic to make its way up the protocol stack to applications running on the system".

CHAPTER 2
REVIEW/BACKGROUND MATERIAL

2.1 Different Types of Firewall:

Firewalls fall into four broad categories: packet filters, circuit level gateways, application level gateways and Stateful multilayer inspection firewalls:

2.1.1 Packet Filtering Firewalls:

Packet filtering firewalls work at the network level of the OSI model, or the IP layer of TCP/IP. They are usually part of a router. A router is a device that receives packets from one network and forwards them to another network. In a packet filtering firewall each packet is compared to a set of criteria before it is forwarded. Depending on the packet and the criteria, the firewall can drop the packet, forward it or send a message to the originator. Rules can include source and destination IP address, source and destination port number and protocol used.

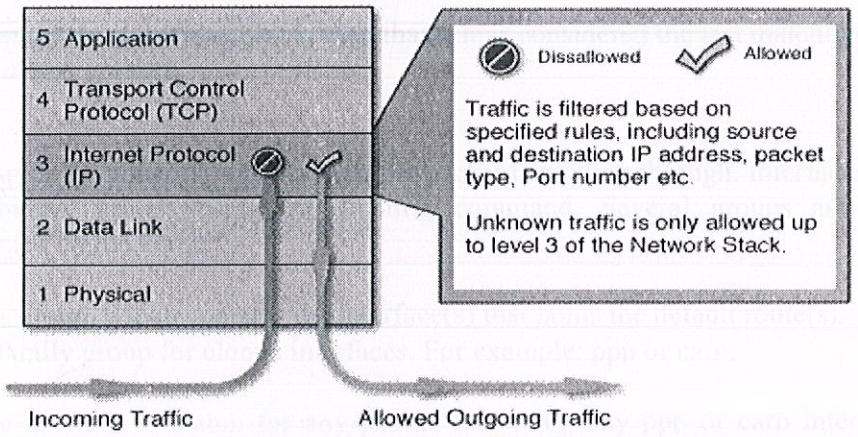

Figure 2. Packet Filter Firewall

The advantage of packet filtering firewalls is their low cost and low impact on network performance. Most routers support packet filtering. Even if other firewalls are used, implementing packet filtering at the router level affords an initial degree of security at a low network layer. This type of firewall only works at the network layer however and does not support sophisticated rule based models. Network Address Translation (NAT) routers offer the advantages of packet filtering firewalls but can also hide the IP addresses of computers behind the firewall, and offer a level of circuit-based filtering.

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

Rule Syntax:

The general, *highly simplified* syntax for filter rules is:

```
action [direction] [log] [quick] [on interface] [af] [proto protocol]
\[from src_addr [port src_port]] [to dst_addr [port dst_port]] \
   [flags tcp_flags] [state] action
```

The action to be taken for matching packets, either pass or block. The pass action will pass the packet back to the kernel for further processing while the block action will react based on the setting of the block policy option. The default reaction may be overridden by specifying either block drop or block return.

Direction:
The direction the packet is moving on an interface, either `in` or `out`.

Log :
Specifies that the packet should be logged via pflogd. If the rule creates state then only the packet which establishes the state is logged. To log all packets regardless, use log (all).

Quick:
If a packet matches a rule specifying quick, then that rule is considered the last matching rule and the specified *action* is taken.

Interface:
The name or group of the network interface that the packet is moving through. Interfaces can be added to arbitrary groups using the ifconfig command. Several groups are also automatically created by the kernel:

- The egress group, which contains the interface(s) that holds the default route(s).
- Interface family group for cloned interfaces. For example: ppp or carp.

This would cause the rule to match for any packet traversing any ppp or carp interface, respectively.

af
The address family of the packet, either inet for IPv4 or inet6 for IPv6. PF is usually able to determine this parameter based on the source and/or destination address(es).
protocol
The Layer 4 protocol of the packet:

- tcp
- udp
- icmp

- icmp6
- A protocol between 0 and 255

src_addr, dst_addr

The source/destination address in the IP header. Addresses can be specified as:

- A single IPv4 or IPv6 address.
- A CIDR network block.
- A fully qualified domain name that will be resolved via DNS when the ruleset is loaded. All resulting IP addresses will be substituted into the rule.
- The name of a network interface or group. Any IP addresses assigned to the interface will be substituted into the rule.
- The name of a network interface followed by */netmask* (i.e., /24). Each IP address on the interface is combined with the netmask to form a CIDR network block which is substituted into the rule.
- The name of a network interface or group in parentheses ( ). This tells PF to update the rule if the IP address(es) on the named interface change. This is useful on an interface that gets its IP address via DHCP or dial-up as the ruleset doesn't have to be reloaded each time the address changes.
- The name of a network interface followed by any one of these modifiers:
  - :network - substitutes the CIDR network block (e.g., 192.168.0.0/24)
  - :broadcast - substitutes the network broadcast address (e.g., 192.168.0.255)
  - :peer - substitutes the peer's IP address on a point-to-point link

  In addition, the :0 modifier can be appended to either an interface name or to any of the above modifiers to indicate that PF should not include aliased IP addresses in the substitution. These modifiers can also be used when the interface is contained in parentheses. Example: fxp0:network:0

- A table
- The keyword urpf-failed can be used for the source address to indicate that it should be run through the uRPF check
- Any of the above but negated using the ! ("not") modifier.
- A set of addresses using a list
- The keyword any meaning all addresses
- The keyword all which is short for from any to any.

src_port, dst_port

The source/destination port in the Layer 4 packet header. Ports can be specified as:

- A number between 1 and 65535
- A valid service name from /etc/services
- A set of ports using a list
- A range:
  - != (not equal)
  - < (less than)
  - > (greater than)
  - <= (less than or equal)
  - >= (greater than or equal)

- >< (range)
- <> (inverse range)

The last two are binary operators (they take two arguments) and do not include the a arguments in the range.

- : (inclusive range)

The inclusive range operator is also a binary operator and does include the arguments in the range.

tcp_flags:

Specifies the flags that must be set in the TCP header when using proto tcp. Flags are specified as flags *check/mask*. For example: flags S/SA - this instructs PF to only look at the S and A (SYN and ACK) flags and to match if only the SYN flag is "on" (and is applied to all TCP rules by default). flags any instructs PF not to check flags.

state

Specifies whether state information is kept on packets matching this rule.

- no state - works with TCP, UDP, and ICMP. PF will not track this connection statefully. For TCP connections, flags any is usually also required.
- keep state - works with TCP, UDP, and ICMP. This option is the default for all filter rules.
- modulate state - works only with TCP. PF will generate strong Initial Sequence Numbers (ISNs) for packets matching this rule.
- synproxy state - proxies incoming TCP connections to help protect servers from spoofed TCP SYN floods. This option includes the functionality of keep state and modulate state.

Default Deny:

The recommended practice when setting up a firewall is to take a "default deny" approach. That is, to deny *everything* and then selectively allow certain traffic through the firewall. This approach is recommended because it errs on the side of caution and also makes writing a ruleset easier.

To create a default deny filter policy, the first two filter rules should be:

```
block in  all
block out all
```

This will block all traffic on all interfaces in either direction from anywhere to anywhere.

Passing Traffic

Traffic must now be explicitly passed through the firewall or it will be dropped bythe default deny policy. This is where packet criteria such as source/destination port, source/destination address, and protocol come into play. Whenever traffic is permitted to pass through the firewall the rule(s) should be written to be as restrictive as possible. This is to ensure that the intended traffic, and only the intended traffic, is permitted to pass.

Some examples:

```
# Pass traffic in on dc0 from the local network, 192.168.0.0/24,
# to the OpenBSD machine's IP address 192.168.0.1. Also, pass the
# return traffic out on dc0.
pass in  on dc0 from 192.168.0.0/24 to 192.168.0.1
pass out on dc0 from 192.168.0.1 to 192.168.0.0/24


# Pass TCP traffic in on fxp0 to the web server running on the
# OpenBSD machine. The interface name, fxp0, is used as the
# destination address so that packets will only match this rule if
# they're destined for the OpenBSD machine.
pass in on fxp0 proto tcp from any to fxp0 port www
```

The quick Keyword

As indicated earlier, each packet is evaluated against the filter ruleset from top to bottom. By default, the packet is marked for passage, which can be changed by any rule, and could be changed back and forth several times before the end of the filter rules. **The last matching rule "wins".** There is an exception to this: The quick option on a filtering rule has the effect of canceling any further rule processing and causes the specified action to be taken. Let's look at a couple examples:

Wrong:

```
block in on fxp0 proto tcp to port ssh
pass  in all
```

In this case, the block line may be evaluated, but will never have any effect, as it is then followed by a line which will pass everything.

Better:

```
block in quick on fxp0 proto tcp to port ssh
pass  in all
```

These rules are evaluated a little differently. If the block line is matched, due to the quick option, the packet will be blocked, and the rest of the ruleset will be ignored.

Keeping State

One of Packet Filter's important abilities is "keeping state" or "stateful inspection". Stateful inspection refers to PF's ability to track the state, or progress, of a network connection. By

storing information about each connection in a state table, PF is able to quickly determine if a packet passing through the firewall belongs to an already established connection. If it does, it is passed through the firewall without going through ruleset evaluation.

Keeping state has many advantages including simpler rulesets and better packet filtering performance. PF is able to match packets moving in *either* direction to state table entries meaning that filter rules which pass returning traffic don't need to be written. And, since packets matching stateful connections don't go through ruleset evaluation, the time PF spends processing those packets can be greatly lessened.

When a rule creates state, the first packet matching the rule creates a "state" between the sender and receiver. Now, not only do packets going from the sender to receiver match the state entry and bypass ruleset evaluation, but so do the reply packets from receiver to sender.

All *pass* rules automatically create a state entry when a packet matches the rule. This can be explicitly disabled by using the no state option.

```
pass out on fxp0 proto tcp from any to any
```

This rule allows any outbound TCP traffic on the fxp0 interface and also permits the reply traffic to pass back through the firewall. Keeping state significantly improves the performance of your firewall as state lookups are dramatically faster than running a packet through the filter rules.

The modulate state option works just like keep state except that it only applies to TCP packets. With modulate state, the Initial Sequence Number (ISN) of outgoing connections is randomized. This is useful for protecting connections initiated by certain operating systems that do a poor job of choosing ISNs. To allow simpler rulesets, the modulate state option can be used in rules that specify protocols other than TCP; in those cases, it is treated as keep state.

Keep state on outgoing TCP, UDP, and ICMP packets and modulate TCP ISNs:

```
pass out on fxp0 proto { tcp, udp, icmp } from any /to any modulate state
```

Another advantage of keeping state is that corresponding ICMP traffic will be passed through the firewall. For example, if a TCP connection passing through the firewall is being tracked statefully and an ICMP source-quench message referring to this TCP connection arrives, it will be matched to the appropriate state entry and passed through the firewall.

The scope of a state entry is controlled globally by the state-policy runtime option and on a per rule basis by the if-bound and floating state option keywords. These per rule keywords have the same meaning as when used with the state-policy option. Example:

```
pass out on fxp0 proto { tcp, udp, icmp } from any \
      to any modulate state (if-bound)
```

This rule would dictate that in order for packets to match the state entry, they must be transiting the `fxp0` interface.

Keeping State for UDP

One will sometimes hear it said that, "One can not create state with UDP as UDP is a stateless protocol!" While it is true that a UDP communication session does not have any concept of state (an explicit start and stop of communications), this does not have any impact on PF's ability to create state for a UDP session. In the case of protocols without "start" and "end" packets, PF simply keeps track of how long it has been since a matching packet has gone through. If the timeout is reached, the state is cleared. The timeout values can be set in the options section of the pf.conf file.

2.1.2 Circuit Level Firewalls:

Circuit level gateways work at the session layer of the OSI model, or the TCP layer of TCP/IP. They monitor TCP handshaking between packets to determine whether a requested session is legitimate. Information passed to remote computer through a circuit level gateway appears to have originated from the gateway. This is useful for hiding information about protected networks.

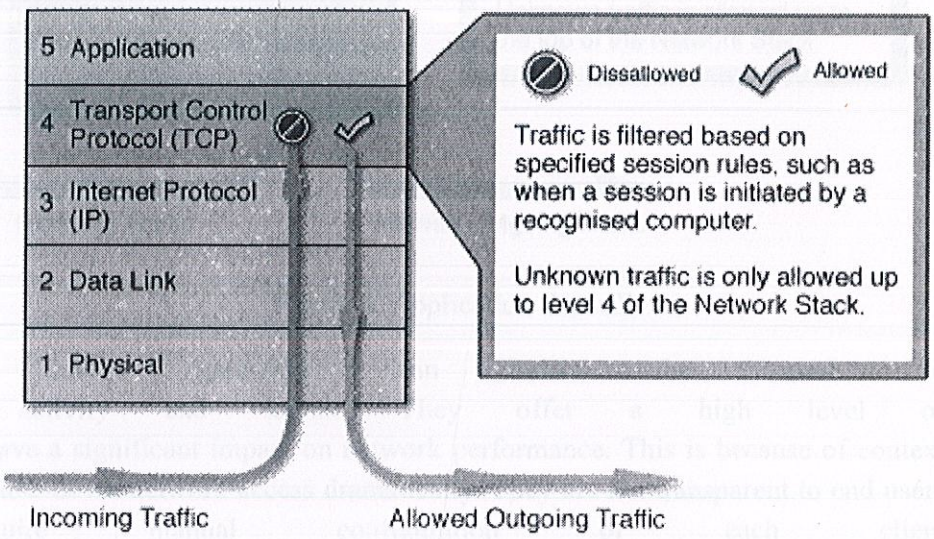

Figure 3. Circuit Level Firewall

Circuit level gateways are relatively inexpensive and have the advantage of hiding information about the private network they protect. On the other hand, they do not filter individual packets.

2.1.3 Application Level Firewalls:

Application level gateways, also called proxies, are similar to circuit-level gateways except that they are application specific. They can filter packets at the application layer of the OSI model. Incoming or outgoing packets cannot access services for which there is no proxy. In plain terms, an application level gateway that is configured to be a web proxy will not allow any ftp, gopher, telnet or other traffic through. Because they examine packets at application layer, they can filter application specific commands such as http:post and get, etc. This cannot be accomplished with either packet filtering firewalls or circuit level neither of which knows anything about the application level information.

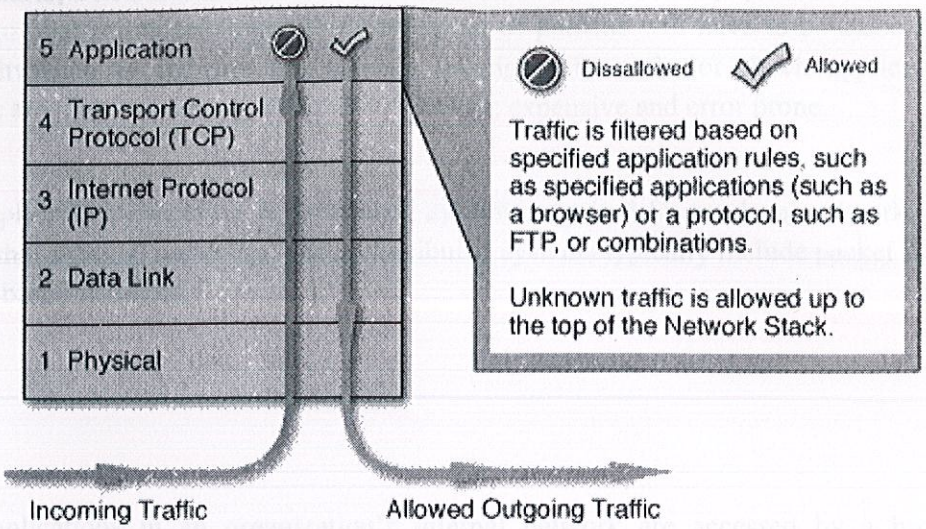

Figure 4. Application firewall

Application level gateways can also be used to log user activity and logins. They offer a high level of security, but have a significant impact on network performance. This is because of context switches that slow down network access dramatically. They are not transparent to end users and require manual configuration of each client computer.

Example

Consider a medical record application in a Hospital. One of the services it provides is to allow patients to lookup their personal medical records from home. To ensure that only patients can access this service, the patient must first be authenticated and then she must be identified as a patient. Finally, the application must ensure that only the medical records belonging to the patient are returned (i.e., match the name in the medical record with that of the user).

One way to provide this security is to let application maintain list of all valid patients with their authentication credentials, and implement the code for blocking unauthorized access from within the application level code of the application.

This approach has several problems. In the future, if the hospital decides to allow patients to be able to schedule appointments, it will have to repeat the implementation of the access control code for the scheduling application as well. Furthermore, if there are changes in hospital business policies, e.g. they want to allow external primary care physicians access the medical records of their own patients, these applications will have to be rewritten. In this

changing scenario, a new access control list for authorized primary care physicians will have to be added to medical record application, and a list of patients will have to be associated with each physician to indicate the patients belonging to a doctor. Such application modifications are time consuming, difficult to manage, expensive and error prone.

Context

Enterprise applications executing in distributed systems accessed from a local network, the Internet, or other external networks. These distributed systems typically include packet filter and/or proxy-based firewalls.

Problem

Enterprise applications in an organization's internal network are accessed by a broad spectrum of users that may attempt to abuse its resources (leakage, modification or destruction of data). These applications can be numerous, thus implementing access control independently in ad hoc ways and may make the system more complex and thus less secure.

Moreover, traditional network firewalls (application layer firewalls or packet filters), do not make it possible to define high level rules (role-based or individual-based rules) that could make the implementation of business security policies easier and simpler.

Forces

- There may be many users (subjects) that need to access an application in different ways; the firewall must adapt to this variety.
- There are many ways to filter application inputs, we need to separate the filtering code from the application code.
- There may be numerous applications that may require different levels of security. We need to define appropriate policies for each application.
- The business policies are constantly changing and they need to be constantly updated; hence it should be easy to change the firewall filtering configuration
- The number of users and applications may increase significantly; adding more users or applications should be done transparently and at proper cost.
- Network firewalls cannot understand the semantics of applications and are unable to filter out potentially harmful messages.

Solution

Interpose a firewall that can analyze incoming requests for application services and check them for authorization. A client can access a service of an application only if a specific policy authorizes it to do so. Policies for each application are centralized within the Application Firewall, and they are accessed through a PolicyAuthorizationPoint. Each

application is accessed by a client through a PolicyEnforcementPoint that enforces access control by looking for a matching policy in the PolicyBase. This enforcement may include authenticating the client through its identity data stored in the IdentityBase.

Dynamics

We describe the dynamic aspects of the Application Firewall using sequence diagrams for two use cases: filtering a Client's request with user authentication and adding a new policy.

Filtering a Client's Request with user authentication:

Summary: A Client requests access to a service of an application to either input or retrieve information. The access request is made through the PolicyEnforcementPoint, which accesses the PolicyAuthorizationPoint to determines whether to accept or deny the request. Figure 3 corresponds to this basic use case.

Actors: A Client

Precondition: Existing IdentityBase and PolicyBase classes must be in place in the firewall. The IdentityBase contains the data necessary to authenticate a Client. The PolicyBase contains specific policies defined by the organization.

Description:

a. A Client requests access to an application.
b. An Application Firewall, through its PolicyEnforcementPoint, intercepts the request and accesses the PolicyAuthorizationPoint.
c. The PolicyAuthorizationPoint authenticates the Client through its IdentityBase. This step may be avoided for each request through the use of a Session class.
d. Once the Client is authenticated and identified, the PolicyAuthorizationPoint filters the request according to the PolicyBase. The request is accepted or denied according to the defined policies.
e. If the request is accepted, the firewall allows access to the service of the application and the access is logged into the Application Firewall.

Alternate Flows:

If the Client is not recognized or if no policy allows the specific Client to access the specified service, the firewall rejects the access request to the service.

If the user has already been authenticated, the Client may not be authenticated again (Single Sign-On use).

Postcondition: The firewall has provided the access of a Client to a service, based on verifying the identity of the Client, and the existence of a matching policy.

Adding a new policy:

Summary: The security administrator intends to add a new policy to the set of policies. Before adding it, the firewall checks whether the new policy to be added does not already exist in the rule set. Figure 4 illustrates this use case.

Actors: Administrator.

Precondition: The administrator must have authorization to add rules.

Description:

a. The administrator initiates the addition of a new rule.
b. If the rule does not already exist in the rule set then it is added.
c. The firewall acknowledges the addition of the new rule.

Alternate Flow: The rule is not added because it already exists in the rule set.

Postcondition: A new rule is added to the rule set of the firewall.

2.1.4 Stateful Firewalls:

Stateful multilayer inspection firewalls combine the aspects of the other three types of firewalls. They filter packets at the network layer, determine whether session packets are legitimate and evaluate contents of packets at the application layer. They allow direct connection between client and host, alleviating the problem caused by the lack of transparency of application level gateways. They rely on algorithms to recognize and process application layer data instead of running application specific proxies. Stateful multilayer inspection firewalls offer a high level of security, good performance and transparency to end users.

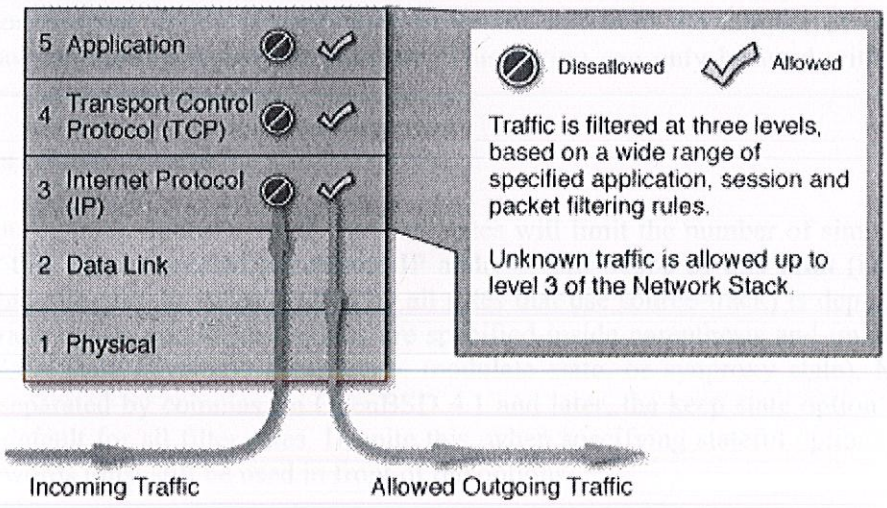

Figure 5. Stateful Firewall

Stateful Tracking Options

Filter rules that create state entries can specify various options to control the behavior of the resulting state entry. The following options are available:

max *number*

Limit the maximum number of state entries the rule can create to *number*. If the maximum is reached, packets that would normally create state fail to match this rule until the number of existing states decreases below the limit.

no state

Prevents the rule from automatically creating a state entry.

source-track

This option enables the tracking of number of states created per source IP address. This option has two formats:

- source-track rule - The maximum number of states created by this rule is limited by the rule's max-src-nodes and max-src-states options. Only state entries created by this particular rule count toward the rule's limits.
- source-track global - The number of states created by all rules that use this option is limited. Each rule can specify different max-src-nodes and max-src-states options, however state entries created by any participating rule count towards each individual rule's limits.

The total number of source IP addresses tracked globally can be controlled via the src-nodes runtime option.

max-src-nodes *number*

When the source-track option is used, max-src-nodes will limit the number of source IP addresses that can simultaneously create state. This option can only be used with source-track rule.

`max-src-states` *number*

When the source-track option is used, max-src-states will limit the number of simultaneous state entries that can be created per source IP address. The scope of this limit (i.e., states created by this rule only or states created by all rules that use source-track) is dependent on the source-track option specified. Options are specified inside parenthesis and immediately after one of the state keywords (keep state, modulate state, or synproxy state). Multiple options are separated by commas. In OpenBSD 4.1 and later, the keep state option became the implicit default for all filter rules. Despite this, when specifying stateful options, one of the state keywords must still be used in front of the options.

An example rule: `pass in on $ext_if proto tcp to $web_server \port www keep state(max 200, source-track rule, max-src-nodes 100, max-src-states 3)`

The rule above defines the following behavior:

- Limit the absolute maximum number of states that this rule can create to 200
- Enable source tracking; limit state creation based on states created by this rule only
- Limit the maximum number of nodes that can simultaneously create state to 100
- Limit the maximum number of simultaneous states per source IP to 3

A separate set of restrictions can be placed on stateful TCP connections that have completed the 3-way handshake.

`max-src-conn` *number*

Limit the maximum number of simultaneous TCP connections which have completed the 3 way handshake that a single host can make.
max-src-conn-rate *number / interval*
Limit the rate of new connections to a certain amount per time interval.

Both of these options automatically invoke the source-track rule option and are incompatible with source-track global.

Since these limits are only being placed on TCP connections that have completed the 3-way handshake, more aggressive actions can be taken on offending IP addresses.

overload <*table*>
Put an offending host's IP address into the named table.
flush [global]
Kill any other states that match this rule and that were created by this source IP. Whenglobal is specified, kill all states matching this source IP, regardless of which rule created the state.

An example:

```
table <abusive_hosts> persist
block in quick from <abusive_hosts>

pass in on $ext_if proto tcp to $web_server \
   port www flags S/SA keep state \
   (max-src-conn 100, max-src-conn-rate 15/5, overload <abusive_hosts> flush)
```

This does the following:

- Limits the maximum number of connections per source to 100
- Rate limits the number of connections to 15 in a 5 second span
- Puts the IP address of any host that breaks these limits into the <abusive_hosts> table
- For any offending IP addresses, flush any states created by this rule.

TCP Flags

Matching TCP packets based on flags is most often used to filter TCP packets that are attempting to open a new connection. The TCP flags and their meanings are listed here:

- **F** : FIN - Finish; end of session
- **S** : SYN - Synchronize; indicates request to start session

- **R** : RST - Reset; drop a connection
- **P** : PUSH - Push; packet is sent immediately
- **A** : ACK - Acknowledgement
- **U** : URG - Urgent
- **E** : ECE - Explicit Congestion Notification Echo
- **W** : CWR - Congestion Window Reduced

To have PF inspect the TCP flags during evaluation of a rule, the flags keyword is used with the following syntax:

flags *check/mask*
flags any

The *mask* part tells PF to only inspect the specified flags and the *check* part specifies which flag(s) must be "on" in the header for a match to occur. Using the any keyword allows any combination of flags to be set in the header.

pass in on fxp0 proto tcp from any to any port ssh flags S/SA
pass in on fxp0 proto tcp from any to any port ssh

As flags S/SA is set by default, the above rules are equivalent, Each of these rules passes TCP traffic with the SYN flag set while only looking at the SYN and ACK flags. A packet with the SYN and ECE flags would match the above rules, while a packet with SYN and ACK or just ACK would not.

The default flags can be overridden by using the flags option as outlined above.

One should be careful with using flags -- understand what you are doing and why, and be careful with the advice people give as a lot of it is bad. Some people have suggested creating state "only if the SYN flag is set and no others". Such a rule would end with:

. . . flags S/FSRPAUEW *bad idea!!*

The theory is, create state only on the start of the TCP session, and the session should start with a SYN flag, and no others. The problem is some sites are starting to use the ECN flag and any site using ECN that tries to connect to you would be rejected by such a rule. A much better guideline is to not specify any flags at all and let PF apply the default flags to your rules. If you truly need to specify flags yourself then this combination should be safe:

. . . flags S/SAFR

While this is practical and safe, it is also unnecessary to check the FIN and RST flags if traffic is also being scrubbed. The scrubbing process will cause PF to drop any incoming packets with illegal TCP flag combinations (such as SYN and RST) and to normalize potentially ambiguous combinations (such as SYN and FIN).

TCP SYN Proxy

Normally when a client initiates a TCP connection to a server, PF will pass the handshake packets between the two endpoints as they arrive. PF has the ability, however, to proxy the handshake. With the handshake proxied, PF itself will complete the handshake with the client, initiate a handshake with the server, and then pass packets between the two. The

benefit of this process is that no packets are sent to the server before the client completes the handshake. This eliminates the threat of spoofed TCP SYN floods affecting the server because a spoofed client connection will be unable to complete the handshake.

The TCP SYN proxy is enabled using the synproxy state keywords in filter rules. Example:

pass in on $ext_if proto tcp to $web_server port www synproxy state

Here, connections to the web server will be TCP proxied by PF.

Because of the way synproxy state works, it also includes the same functionality as keep state and modulate state.

The SYN proxy will not work if PF is running on a bridge(4).

Blocking Spoofed Packets

Address "spoofing" is when an malicious user fakes the source IP address in packets they transmit in order to either hide their real address or to impersonate another node on the network. Once the user has spoofed their address they can launch a network attack without revealing the true source of the attack or attempt to gain access to network services that are restricted to certain IP addresses.

PF offers some protection against address spoofing through the antispoof keyword:

antispoof [log] [quick] for *interface* [*af*]

log

Specifies that matching packets should be logged via pflogd.

quick

If a packet matches this rule then it will be considered the "winning" rule and ruleset evaluation will stop.

interface

The network interface to activate spoofing protection on. This can also be a list of interfaces.

af

The address family to activate spoofing protection for, either inet for IPv4 or inet6 for IPv6.

Example:

antispoof for fxp0 inet

When a ruleset is loaded, any occurrences of the antispoof keyword are expanded into two filter rules. Assuming that interface fxp0 has IP address 10.0.0.1 and a subnet mask of 255.255.255.0 (i.e., a /24), the above antispoof rule would expand to:

```
block in on ! fxp0 inet from 10.0.0.0/24 to any
block in inet from 10.0.0.1 to any
```

These rules accomplish two things:

- Blocks all traffic coming from the 10.0.0.0/24 network that does *not* pass in through fxp0. Since the 10.0.0.0/24 network is on the fxp0 interface, packets with a source address in that network block should never be seen coming in on any other interface.
- Blocks all incoming traffic from 10.0.0.1, the IP address on fxp0. The host machine should never send packets to itself through an external interface, so any incoming packets with a source address belonging to the machine can be considered malicious.

NOTE: The filter rules that the antispoof rule expands to will also block packets sent over the loopback interface to local addresses. It's best practice to skip filtering on loopback interfaces anyways, but this becomes a necessity when using antispoof rules:

```
set skip on lo0

antispoof for fxp0 inet
```

Usage of antispoof should be restricted to interfaces that have been assigned an IP address. Using antispoof on an interface without an IP address will result in filter rules such as:

```
block drop in on ! fxp0 inet all
block drop in inet all
```

With these rules there is a risk of blocking *all* inbound traffic on *all* interfaces.

Unicast Reverse Path Forwarding

PF offers a Unicast Reverse Path Forwarding (uRPF) feature. When a packet is run through the uRPF check, the source IP address of the packet is looked up in the routing table. If the outbound interface found in the routing table entry is the same as the interface that the packet just came in on, then the uRPF check passes. If the interfaces don't match, then it's possible the packet has had its source address spoofed.

The uRPF check can be performed on packets by using the urpf-failed keyword in filter rules:

```
block in quick from urpf-failed label uRPF
```

Note that the uRPF check only makes sense in an environment where routing is symmetric.

uRPF provides the same functionality as antispoof rules.

Passive Operating System Fingerprinting

Passive OS Fingerprinting (OSFP) is a method for passively detecting the operating system of a remote host based on certain characteristics within that host's TCP SYN packets. This information can then be used as criteria within filter rules.

PF determines the remote operating system by comparing characteristics of a TCP SYN packet against the fingerprints file, which by default is /etc/pf.os. Once PF is enabled, the current fingerprint list can be viewed with this command:

pfctl -s osfp

Within a filter rule, a fingerprint may be specified by OS class, version, or subtype/patch level. Each of these items is listed in the output of the pfctl command shown above. To specify a fingerprint in a filter rule, the os keyword is used:

```
pass  in on $ext_if from any os OpenBSD keep state
block in on $ext_if from any os "Windows 2000"
block in on $ext_if from any os "Linux 2.4 ts"
block in on $ext_if from any os unknown
```

The special operating system class unknown allows for matching packets when the OS fingerprint is not known.

TAKE NOTE of the following:

- Operating system fingerprints are occasionally wrong due to spoofed and/or crafted packets that are made to look like they originated from a specific operating system.
- Certain revisions or patchlevels of an operating system may change the stack's behavior and cause it to either not match what's in the fingerprints file or to match another entry altogether.

- OSFP only works on the TCP SYN packet; it will not work on other protocols or on already established connections.

IP Options

By default, PF blocks packets with IP options set. This can make the job more difficult for "OS fingerprinting" utilities like nmap. If you have an application that requires the passing of these packets, such as multicast or IGMP, you can use the allow-opts directive:

```
pass in quick on fxp0 all allow-opts
```

Filtering Ruleset Example

Below is an example of a filtering ruleset. The machine running PF is acting as a firewall between a small, internal network and the Internet. Only the filter rules are shown; queueing, nat, rdr, etc., have been left out of this example.

```
ext_if = "fxp0"
```

```
int_if = "dc0"
lan_net = "192.168.0.0/24"

# table containing all IP addresses assigned to the firewall
table <firewall> const { self }

# don't filter on the loopback interface
set skip on lo0

# scrub incoming packets
match in all scrub (no-df)

# setup a default deny policy
block all

# activate spoofing protection for all interfaces
block in quick from urpf-failed

# only allow ssh connections from the local network if it's from the
# trusted computer, 192.168.0.15. use "block return" so that a TCP RST is
# sent to close blocked connections right away. use "quick" so that this
# rule is not overridden by the "pass" rules below.
block return in quick on $int_if proto tcp from ! 192.168.0.15 \
  to $int_if port ssh

# pass all traffic to and from the local network.
# these rules will create state entries due to the default
# "keep state" option which will automatically be applied.
pass in  on $int_if from $lan_net
pass out on $int_if to $lan_net

# pass tcp, udp, and icmp out on the external (Internet) interface.
# tcp connections will be modulated, udp/icmp will be tracked
# statefully.
pass out on $ext_if proto { tcp udp icmp } all modulate state

# allow ssh connections in on the external interface as long as they're
# NOT destined for the firewall (i.e., they're destined for a machine on
# the local network). log the initial packet so that we can later tell
# who is trying to connect. use the tcp syn proxy to proxy the connection.
# the default flags "S/SA" will automatically be applied to the rule by
# PF.
pass in log on $ext_if proto tcp to ! <firewall> \
  port ssh synproxy state
```

CHAPTER 3
PROCESS DESCRIPTION

3.1 DATA FLOW DIAGRAMS:

3.1.1 Level 0 DFD:

Figure 6. Level 0 DFD

3.1.2 Level 1 DFD:

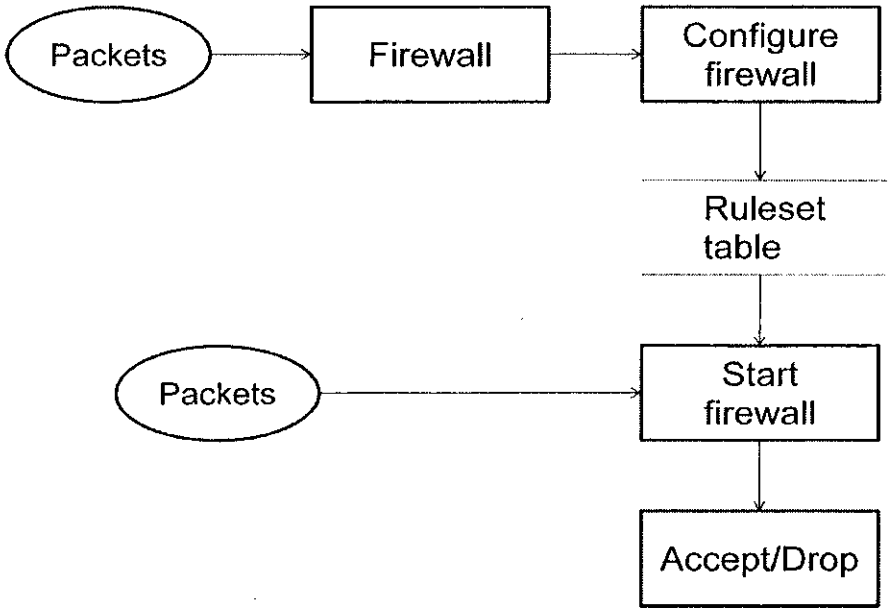

Figure 7. Level 1 DFD

3.1.3 Level 2 DFD:

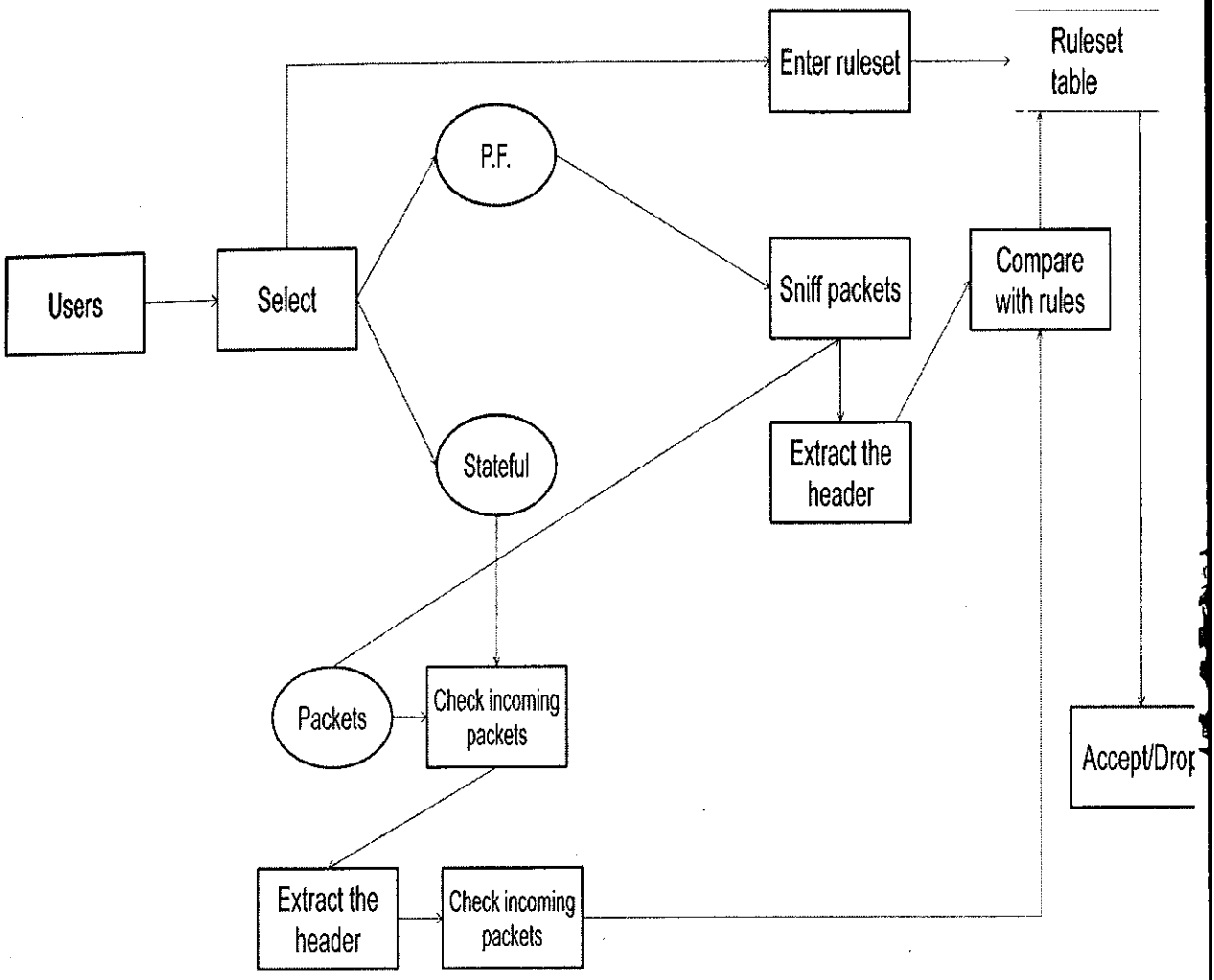

Figure 8. Level 2 DFD

3.2 FLOWCHARTS:

3.2.1 Packet Filtering:

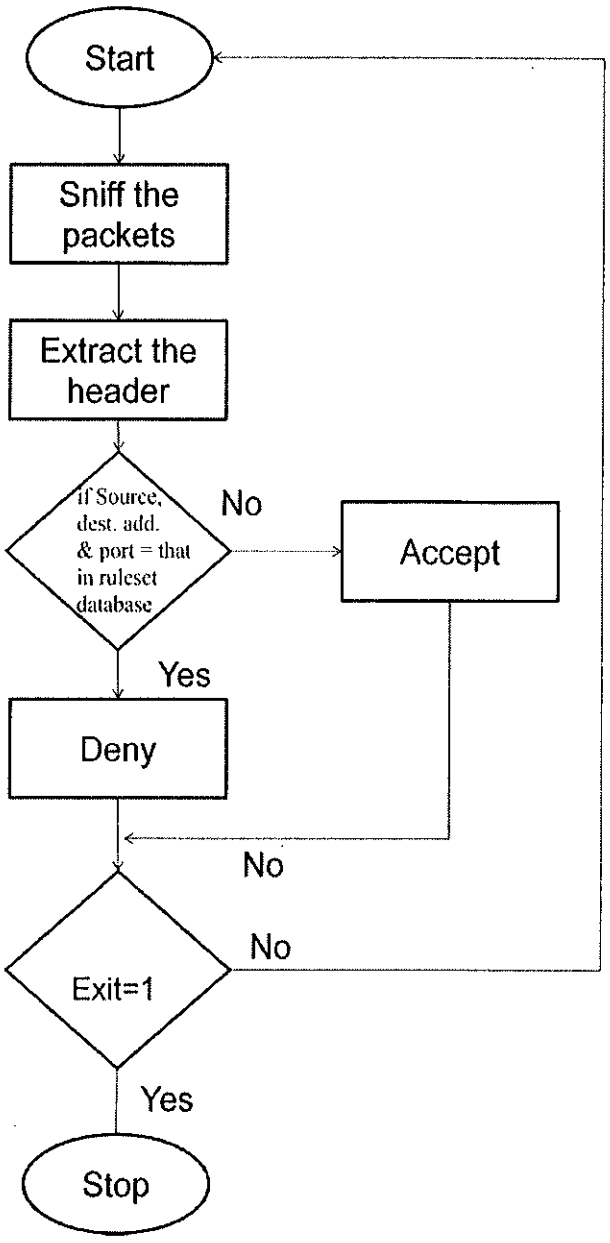

Figure 9. Flowchart 1

3.2.1 Stateful Filtering:

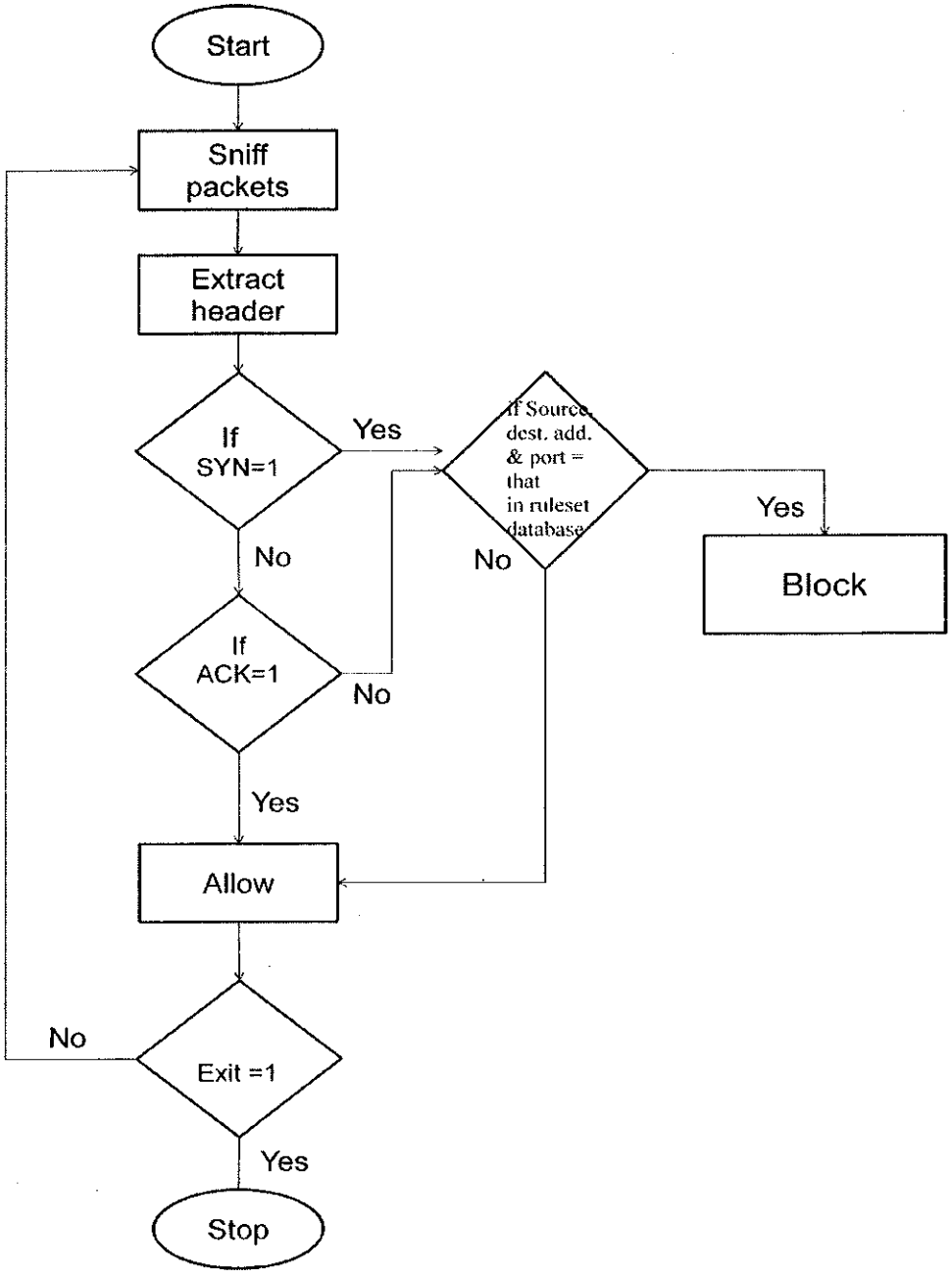

Figure 10. Flowchart 2

3.3 What is IP tables:

Originally, the most popular firewall/NAT package running on Linux was ipchains, but it had a number of shortcomings. To rectify this, the Netfilter organization decided to create a new product

called iptables, giving it such improvements as:

- Better integration with the Linux kernel with the capability of loading iptables-specific kernel modules designed for improved speed and reliability.
- Stateful packet inspection. This means that the firewall keeps track of each connection passing through it and in certain cases will view the contents of data flows in an attempt to anticipate the next action of certain protocols. This is an important feature in the support of active FTP and DNS, as well as many other network services.
- Filtering packets based on a MAC address and the values of the flags in the TCP header. This is helpful in preventing attacks using malformed packets and in restricting access from locally attached servers to other networks in spite of their IP addresses.
- System logging that provides the option of adjusting the level of detail of the reporting.
- Better network address translation.
- Support for transparent integration with such Web proxy programs as Squid.
- A rate limiting feature that helps iptables block some types of denial of service (DoS) attacks.

3.3.1 Packet Processing in IP tables:

All packets inspected by iptables pass through a sequence of built-in tables (queues) for processing. Each of these queues is dedicated to a particular type of packet activity and is controlled by an associated packet transformation/filtering chain.

There are three tables in total. The first is the mangle table which is responsible for the alteration of quality of service bits in the TCP header. This is hardly used in a home or SOHO environment.

The second table is the filter queue which is responsible for packet filtering. It has three built-in chains in which you can place your firewall policy rules. These are the:

- Forward chain: Filters packets to servers protected by the firewall.
- Input chain: Filters packets destined for the firewall.
- Output chain: Filters packets originating from the firewall.

The third table is the nat queue which is responsible for network address translation. It has two built-in chains; these are:

- Pre-routing chain: NATs packets when the destination address of the packet needs to be changed.
- Post-routing chain: NATs packets when the source address of the packet needs to be changed

Queue Type	Queue Function	Packet Transformation Chain in Queue	Chain Function
Filter	Packet filtering	FORWARD	Filters packets to servers accessible by another NIC on the firewall.
		INPUT	Filters packets destined to the firewall.
		OUTPUT	Filters packets originating from the firewall
Nat	Network Address Translation	PREROUTING	Address translation occurs before routing. Facilitates the transformation of the destination IP address to be compatible with the firewall's routing table. Used with NAT of the destination IP address, also known as **destination NAT** or **DNAT**.
		POSTROUTING	Address translation occurs after routing. This implies that there was no need to modify the destination IP address of the packet as in pre-routing. Used with NAT of the source IP address using either one-to-one or many-to-one NAT. This is known as **source NAT**, or **SNAT**.
		OUTPUT	Network address translation for packets generated by the firewall. (Rarely used in SOHO environments)
Mangle	TCP header modification	PREROUTING POSTROUTING OUTPUT INPUT FORWARD	Modification of the TCP packet quality of service bits before routing occurs. (Rarely used in SOHO environments)

Table 1. Processing for packets routed by the firewall

You need to specify the table and the chain for each firewall rule you create. There is an exception: Most rules are related to filtering, so iptables assumes that any chain that's defined without an associated table will be a part of the filter table. The filter table is therefore the default.

To help make this clearer, take a look at the way packets are handled by iptables. In Figure a TCP packet from the Internet arrives at the firewall's interface on Network A to create a data connection.

The packet is first examined by your rules in the mangle table's PREROUTING chain, if any. It is then inspected by the rules in the nat table's PREROUTING chain to see whether the packet requires DNAT. It is then routed.

If the packet is destined for a protected network, then it is filtered by the rules in the FORWARD chain of the filter table and, if necessary, the packet undergoes SNAT in the POSTROUTING chain before arriving at Network B. When the destination server decides to reply, the packet undergoes

the same sequence of steps. Both the FORWARD and POSTROUTING chains may be configured to implement quality of service (QoS) features in their mangle tables, but this is not usually done in SOHO environments.

If the packet is destined for the firewall itself, then it passes through the mangle table of the INPUT chain, if configured, before being filtered by the rules in the INPUT chain of the filter table before. If it successfully passes these tests then it is processed by the intended application on the firewall.

At some point, the firewall needs to reply. This reply is routed and inspected by the rules in the OUTPUT chain of the mangle table, if any. Next, the rules in the OUTPUT chain of the nat table determine whether DNAT is required and the rules in the OUTPUT chain of the filter table are then inspected to help restrict unauthorized packets. Finally, before the packet is sent back to the Internet, SNAT and QoS mangling is done by the POSTROUTING chain.

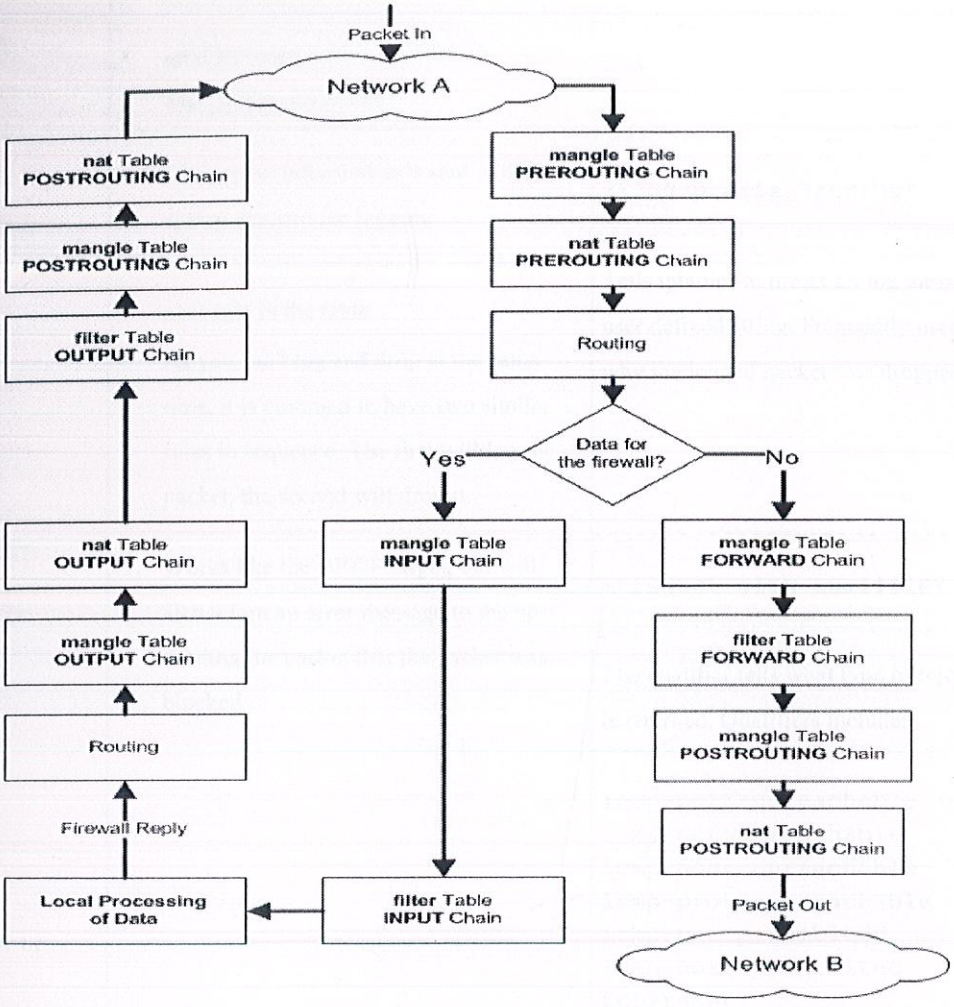

Figure 11. IP tables packet flow diagram

3.3.2 Targets and Jumps:

Each firewall rule inspects each IP packet and then tries to identify it as the target of some sort of operation. Once a target is identified, the packet needs to jump over to it for further processing. Table lists the built-in targets that iptables uses.

target	Desciption	Most Common Options
ACCEPT	▪ iptables stops further processing. ▪ The packet is handed over to the end application or the operating system for	N/A

	processing	
DROP	▪ iptables stops further processing. ▪ The packet is blocked	N/A
LOG	▪ The packet information is sent to the syslog daemon for logging ▪ iptables continues processing with the next rule in the table ▪ As you can't log and drop at the same time, it is common to have two similar rules in sequence. The first will log the packet, the second will drop it.	`--log-prefix "string"` Tells iptables to prefix all log messages with a user defined string. Frequently used to tell why the logged packet was dropped

REJECT	▪ Works like the DROP target, but will also return an error message to the host sending the packet that the packet was blocked	`--reject-with qualifier`
		The qualifier tells what type of reject message is returned. Qualifiers include:
		`icmp-port-unreachable (default)`
		`icmp-net-unreachable`
		`icmp-host-unreachable`
		`icmp-proto-unreachable`
		`icmp-net-prohibited`
		`icmp-host-prohibited`
		`tcp-reset`
		`echo-reply`

DNAT	▪ Used to do **destination network address translation**. ie. rewriting the destination IP address of the packet	`--to-destination ipaddress` Tells iptables what the destination IP address should be
SNAT	▪ Used to do **source network address translation** rewriting the source IP address of the packet ▪ The source IP address is user defined	`--to-source <address>[-<address>][:<port>-<port>]` Specifies the source IP address and ports to be used by SNAT.
MASQUERADE	▪ Used to do Source Network Address Translation.	`[--to-ports <port>[-<port>]]`

	▪ By default the source IP address is the same as that used by the firewall's interface	Specifies the range of source ports to which the original source port can be mapped.

Table 2. Most Commonly used targets

3.3.4 Match Criteria:

Each line of an iptables script not only has a jump, but they also have a number of command line options that are used to append rules to chains that match your defined packet characteristics, such the source IP address and TCP port.

iptables command Switch	Desciption
`-t <-table->`	If you don't specify a table, then the `filter` table is assumed. As discussed before, the possible built-in tables include: filter, nat, mangle
`-j <target>`	Jump to the specified target chain when the packet matches the current rule.
`-A`	Append rule to end of a chain
`-F`	Flush. Deletes all the rules in the selected table
`-p <protocol-type>`	Match protocol. Types include, icmp, tcp, udp, and all
`-s <ip-address>`	Match source IP address
`-d <ip-address>`	Match destination IP address
`-i <interface-name>`	Match "input" interface on which the packet enters.
`-o <interface-name>`	Match "output" interface on which the packet exits

Table 3. General Match criteria

In this command switches example

```
iptables -A INPUT -s 0/0 -i eth0 -d 192.168.1.1  -p TCP -j ACCEPT
```

iptables is being configured to allow the firewall to accept TCP packets coming in on interface eth0 from any IP address destined for the firewall's IP address of 192.168.1.1. The 0/0 representation of an IP address means any.

Switch	Desciption
`-p tcp --sport <port>`	TCP source port. Can be a single value or a range in the format: *start-port-number:end-port-number*
`-p tcp --dport <port>`	TCP destination port. Can be a single value or a range in the format: *starting-port:ending-port*
`-p tcp --syn`	Used to identify a new TCP connection request. ! --syn means, not a new connection request
`-p udp --sport <port>`	UDP source port. Can be a single value or a range in the format: *starting-port:ending-port*
`-p udp --dport <port>`	UDP destination port. Can be a single value or a range in the format: *starting-port:ending-port*

Table 4. Common TCP and UDP match criteria

In this example:

```
iptables -A FORWARD -s 0/0 -i eth0 -d 192.168.1.58 -o eth1 -p TCP \
        --sport 1024:65535 --dport 80 -j ACCEPT
```

iptables is being configured to allow the firewall to accept TCP packets for routing when they enter on interface eth0 from any IP address and are destined for an IP address of 192.168.1.58 that is reachable via interface eth1. The source port is in the range 1024 to 65535 and the destination port is port 80 (www/http).

Matches used with ---icmp-type	Desciption
`--icmp-type <type>`	The most commonly used types are echo-reply and echo-request

Table 5. Common ICMP Match criteria

In this example:

```
iptables -A OUTPUT -p icmp --icmp-type echo-request -j ACCEPT
iptables -A INPUT  -p icmp --icmp-type echo-reply   -j ACCEPT
```

iptables is being configured to allow the firewall to send ICMP echo-requests (pings) and in turn, accept the expected ICMP echo-replies.

Consider another example

```
iptables -A INPUT -p icmp --icmp-type echo-request \
         -m limit --limit 1/s -i eth0 -j ACCEPT
```

The limit feature in iptables specifies the maximum average number of matches to allow per second. You can specify time intervals in the format /second, /minute, /hour, or /day, or you can use abbreviations so that 3/second is the same as 3/s.

In this example, ICMP echo requests are restricted to no more than one per second. When tuned correctly, this feature allows you to filter unusually high volumes of traffic that characterize denial of service (DOS) attacks and Internet worms.

```
iptables -A INPUT -p tcp --syn -m limit --limit 5/s -i eth0 -j ACCEPT
```

You can expand on the limit feature of iptables to reduce your vulnerability to certain types of denial of service attack. Here a defense for SYN flood attacks was created by limiting the acceptance of TCP segments with the SYN bit set to no more than five per second.

Switch	Desciption
`-m multiport --sports <port, port>`	A variety of TCP/UDP source ports separated by commas. Unlike when -m isn't used, they do not have to be within a range.

`-m multiport --dports <port, port>`	A variety of TCP/UDP destination ports separated by commas. Unlike when -m isn't used, they do not have to be within a range.
`-m multiport --ports <port, port>`	A variety of TCP/UDP ports separated by commas. Source and destination ports are assumed to be the same and they do not have to be within a range.
`-m --state <state>`	The most frequently tested states are: **ESTABLISHED:** The packet is part of a connection that has seen packets in both directions **NEW:** The packet is the start of a new connection **RELATED:** The packet is starting a new secondary connection. This is a common feature of such protocols such as an FTP data transfer, or an ICMP error. **INVALID:** The packet couldn't be identified. Could be due to insufficient system resources, or ICMP errors that don't match an existing data flow.

Table 6. Common Extended match criteria

This is an expansion on the previous example:

```
iptables -A FORWARD -s 0/0 -i eth0 -d 192.168.1.58 -o eth1 -p TCP \
        --sport 1024:65535 -m multiport --dports 80,443 -j ACCEPT

iptables -A FORWARD -d 0/0 -o eth0 -s 192.168.1.58 -i eth1 -p TCP \
        -m state --state ESTABLISHED -j ACCEPT
```

Here iptables is being configured to allow the firewall to accept TCP packets to be routed when they enter on interface eth0 from any IP address destined for IP address of 192.168.1.58 that is reachable via interface eth1. The source port is in the range 1024 to 65535 and the destination ports are port 80 (www/http) and 443 (https). The return packets from 192.168.1.58 are allowed to be accepted too. Instead of stating the source and destination ports, you can simply allow packets related to established connections using the -m state and --state ESTABLISHED options.

Using User Defined Chains:

As you may remember, you can configure iptables to have user-defined chains. This feature is frequently used to help streamline the processing of packets. For example, instead of using

a single, built-in chain for all protocols, you can use the chain to determine the protocol type for the packet and then hand off the actual final processing to a user-defined, protocol-specific chain in the filter table. In other words, you can replace a long chain with a stubby main chain pointing to multiple stubby chains, thereby shortening the total length of all chains the packet has to pass through. For example

```
iptables -A INPUT -i eth0  -d 206.229.110.2 -j fast-input-queue
iptables -A OUTPUT -o eth0 -s 206.229.110.2 -j fast-output-queue

iptables -A fast-input-queue  -p icmp -j icmp-queue-in
iptables -A fast-output-queue -p icmp -j icmp-queue-out

iptables -A icmp-queue-out -p icmp --icmp-type echo-request \
         -m state --state NEW -j ACCEPT

iptables -A icmp-queue-in -p icmp --icmp-type echo-reply -j ACCEPT
```

Here six queues help assist in improving processing speed. Table summarizes the function of each.

Chain	Desciption
`INPUT`	The regular built-in INPUT chain in iptables
`OUTPUT`	The regular built-in OUTPUT chain in `iptables`
`fast-input-queue`	Input chain dedicated to identifying specific protocols and shunting the packets to protocol specific chains.
`fast-output-queue`	Output chain dedicated to identifying specific protocols and shunting the packets to protocol specific chains.
`icmp-queue-out`	Output queue dedicated to ICMP
`icmp-queue-in`	Input queue dedicated to ICMP

Table 7. Custom Queues Example Listing

3.4 Hardware Requirements

Server Side:

- Operating System: LINUX or UNIX
- Processor: Pentium 3.0 GHz or higher
- RAM: 256 Mb or more
- Hard Drive: 1 GB or more

Client side:

- Operating System: LINUX or UNIX.
- Processor: Pentium III or 2.0 GHz or higher.
- RAM: 256 Mb or more

3.5 Software Requirements

- ***Programming Language:*** Shell programming
- ***Background:*** Linux

CHAPTER 4
SAMPLE CODES

4.1 Init File:

```
#!/bin/sh
test -x /sbin/iptables || exit 0
set –e
# location of the rules
RULESDIR=/etc/fw/rules.d
MODULES=/etc/fw/modules
case "$1" in
 start)
  echo -n "Init firewall: "
  # include configs
  . /etc/fw/aliases
  # load extensions
  while read MODULE ; do
       case "$MODULE" in
            ""|\#*) continue ;;
       esac
       modprobe $MODULE
  done < ${MODULES}
  # install the rules
  for RULES in $RULESDIR/* ; do
   case "$RULES" in
       *~) continue ;;
```

```
    esac
    # echo -n "`basename $RULES | sed -e s/.._.*/.*/g` "
    echo -n "`basename $RULES` "
    while read RULE ; do
        case "$RULE" in
                ""|\#*) continue ;;
        esac
        case "$RULE" in
                ""|\-*)
                # iptables rule
                # echo "iptables: $RULE"
                eval iptables $RULE
                continue
                ;;
        esac
        eval $RULE
    done < ${RULES}
  done
  echo "done."
  ;;
stop)
  echo -n "Clear firewall: "
  iptables --flush
  iptables --delete-chain
  iptables --policy INPUT ACCEPT
  iptables --policy FORWARD DROP
```

```
    iptables --policy OUTPUT ACCEPT
    # unload extensions
    while read MODULE ; do
        case "$MODULE" in
            ""|\#*) continue ;;
        esac
        rmmod $MODULE
    done < ${MODULES}
    echo "done."
    echo "Caution: firewall is open now!"
    ;;
  test)
    echo "Test firewall-rules: "
    $0 start
    echo "Rules now active for 30 seconds ..."
    sleep 30
    $0 stop
    echo "Caution rules now deactivated !"
    ;;
  show)
    echo "Active firewall-rules: "
    iptables -L $2
    ;;
  *)
    echo "Usage: /etc/init.d/fw {start|stop|test|show|reload}"
    exit 1
```

```
esac

exit 0
```

4.2 Modules:

```
#
# modules for fw
#


# load needed extensions

ip_tables

iptable_filter

ipt_LOG

ipt_REJECT

ipt_owner

ipt_limit

# connection tracking (stateful)

ip_conntrack

ipt_state

# for ftp

ip_conntrack_ftp

# NAT

iptable_nat

ipt_MASQUERADE

ipt_REDIRECT

# for passive ftp

ip_nat_ftp
```

4.3 Rules:

```
00_init::

#
# filter-rules for fw
#

# init

#

# flush tables

--flush

-t nat --flush

--delete-chain

# temp default policy

--policy INPUT                  ACCEPT

--policy FORWARD    DROP

--policy OUTPUT                 ACCEPT

#

05_targets::

#
# targets for rules
#

#
# Drop: drop bad packets
#

R=Drop

-N $R

-A $R   -m limit --limit 3/second        -j LOG --log-prefix "$R "

# -A $R          -j REJECT
```

```
-A $R  -j DROP

#
# Accept: accept good packets
#

R=Accept

-N $R

-A $R  -m limit --limit 3/second       -j LOG --log-prefix "$R "

-A $R  -j ACCEPT

#
# Accept: target for accepted established packets
#

R=AcceptEstablished

-N $R

#-A $R -m limit --limit 3/second       -j LOG --log-prefix "$R "

-A $R  -j ACCEPT

#
# Accept: target for accepted related packets
#

R=AcceptRelated

-N $R

-A $R  -m limit --limit 3/second       -j LOG --log-prefix "$R "

-A $R  -j ACCEPT

#
# Accept: target for new packets of established connections
#

R=AcceptNew

-N $R

-A $R  -m limit --limit 3/second       -j LOG --log-prefix "$R "

-A $R  -j ACCEPT
```

10_antispoof::

#
 # Chunk: deny connections from or to abused/spoofed addresses
#

R=Junk

-N $R

 # junk

-A $R -i $EXT_IF -d $EVERY_BCAST -j DROP

-A $R -i $INT_IF -d $EVERY_BCAST -j DROP

-A $R -i $EXT_IF -d $EXT_BCAST -j DROP

-A $R -i $INT_IF -d $INT_BCAST -j DROP

 # netbios-bc

-A $R -i $EXT_IF -d $INT_BCAST -p udp --dport 137:139 -j

DROP

-A $R -i $INT_IF -d $INT_BCAST -p udp --dport 137:139 -j

DROP

-A $R -i $DMZ_IF -d $DMZ_BCAST -p udp --dport 137:139 -j

DROP

 # spoofing

-A $R -i $EXT_IF -s $LOC_NET -j Drop

-A $R -i $EXT_IF -s $EXT_IP -j Drop

-A $R -i $EXT_IF -s $INT_NET -j Drop

-A $R -i $INT_IF -s $LOC_NET -j Drop

-A $R -i $INT_IF -s $INT_IP -j Drop

 # fixme: -s in
 # -t nat -A PREROUTING
#

```
20_stateful::

#
# stateful connection tracking
#

#
# New: make a new connection and accept
#

R=New

-N $R

#-A $R -j LOG --log-prefix "$R "

# In the light of the fact that ACK packets can create state table entries,

# the following contribution from Henrik Nordstrum is insightful: To make sure

# that NEW tcp connections are packets with SYN set, use the following rule:

-A $R -m state --state NEW    -p tcp ! --syn    -j LOG --log-prefix "$R not syn:"

-A $R -m state --state NEW    -p tcp ! --syn    -j Drop

# Note that doing this will prevent idle sessions from continuing once they

# have expired from the conntrack table. In the normal "relaxed" view such

# connections initiated from the correct direction (i.e. the direction you

# allow NEW packets through) can normally continue even if expired from

# conntrack, provided that the first data/ack packet that resumes the

# connection comes from the correct direction.

-A $R -m state --state NEW                    -j AcceptNew

#
# Established: check for established connections
#

R=Established

-N $R

#-A $R -j LOG --log-prefix "$R "

# if packet is related to a establised connection accept
```

```
#-A $R -m state --state ESTABLISHED        -p udp  -j AcceptEstablishedUDP

#-A $R -m state --state ESTABLISHED        -p icmp         -j AcceptEstablishedICMP

-A $R -m state --state ESTABLISHED                  -j AcceptEstablished

-A $R -m state --state RELATED                      -j AcceptRelated


30_fw-intern::

#
# fw intern services
#

##
## FW-FW: helper connections from fw to fw
##
## input chain
##

R=FW-FW_INPUT

-N $R

#-A $R          -j LOG --log-prefix "$R "

# allow local lookups on local interface

-A $R -i $LOC_IF -s $LOC_NET      -d $LOC_NET          -j ACCEPT

-A $R -i $LOC_IF -s $INT_IP       -d $INT_IP           -j Accept

-A $R -i $LOC_IF -s $EXT_IP       -d $EXT_IP           -j Accept

# allow local lookups in intern interface

-A $R -i $INT_IF -s $INT_IP -d $INT_IP              -j Accept

##
## output chain
##

R=FW-FW_OUTPUT

-N $R

#-A $R          -j LOG --log-prefix "$R "

# allow local lookups on local interface
```

```
-A $R -o $LOC_IF -s $LOC_NET    -d $LOC_NET        -j ACCEPT

-A $R -o $LOC_IF -s $INT_IP     -d $INT_IP         -j Accept

-A $R -o $LOC_IF -s $EXT_IP     -d $EXT_IP         -j Accept

# allow local lookups in intern interface

-A $R -o $INT_IF -s $INT_IP -d $INT_IP             -j Accept

#

40_fw-services::

#
# services on the fw for intern net
#

##
## Int-FW: allow connections from intern to fw
##

R=Int-FW

-N $R

#-A $R -j LOG --log-prefix "$R "

#
# allow ping from intern net
#

-A $R -j New -i $INT_IF -s $INT_NET             -p icmp             --icmp-type ping

#
# allow domain-request from all intern networks
#

-A $R -j New -i $INT_IF -s $INT_NET             -p udp          --dport domain

#
# allow dhcp/bootp-request from intern network

#

-A $R -j New -i $INT_IF -s $INT_NET             -p udp  --sport bootpc --dport bootps

#
# allow ssh for service from intern support
#
```

```
-A $R -j New -i $INT_IF -s $INT_SUPPORT          -p tcp --syn      --dport ssh

#
# allow smtp from intern mailserver
#

-A $R -j New -i $INT_IF -s $INT_SMTP1           -p tcp --syn      --dport smtp

-A $R -j New -i $INT_IF -s $INT_SMTP2           -p tcp --syn      --dport smtp

#
# allow webproxy from intern net
#

-A $R -j New -i $INT_IF -s $INT_NET             -p tcp --syn      --dport webcache

#
# reject auth on intern interface
#

-A $R -j REJECT -i $INT_IF                      -p tcp --syn      --dport auth --reject-with port-
unreach

##
## FW-Int: allow connections from the fw to intern
##

R=FW-Int

-N $R

#-A $R -j LOG --log-prefix "$R "

#
# allow ping to intern net
#

-A $R -j New -o $INT_IF             -d $INT_NET  -p icmp              --icmp-type ping

#
# allow dhcp/bootp-replies from fw to intern network
#

-A $R -j New -o $INT_IF             -d $INT_NET  -p udp          --sport bootps --dport

bootpc

#
# allow smtp to intern mailserver
#

-A $R -j New -o $INT_IF             -d $INT_SMTP1       -p tcp --syn      --dport smtp
```

```
-A $R -j New -o $INT_IF          -d $INT_SMTP2      -p tcp --syn    --dport smtp

#
# allow auth (when incomming smtp) to intern mailserver
#

-A $R -j New -o $INT_IF      -d $INT_SMTP1      -p tcp --syn    --dport auth

-A $R -j New -o $INT_IF      -d $INT_SMTP2      -p tcp --syn    --dport auth

#
# alternativ auto-reject auth
#

#-A $R -o $INT_IF -p tcp --syn -d $INT_SMTP --dport auth -j REJECT --reject-with port-
unreach

#
# allow ssh to intern net
#

-A $R -j New -o $INT_IF          -d $INT_NET  -p tcp --syn    --dport ssh


43_fw-services-dmz::

#
# services on the fw for dmz net
#

##
## DMZ-FW: allow connections from dmz net to fw
##

R=DMZ-FW

-N $R

#-A $R -j LOG --log-prefix "$R "

#
# allow ping from dmz net
#

-A $R -j New -i $DMZ_IF -s $DMZ_NET          -p icmp              --icmp-type ping

#
# allow domain-request from dmz net
#

-A $R -j New -i $DMZ_IF -s $DMZ_NET          -p udp          --dport domain
```

```
#
# allow smtp from dmz net
#

-A $R -j New -i $DMZ_IF -s $DMZ_NET              -p tcp --syn     --dport smtp

#
# allow webproxy from dmz net
#

-A $R -j New -i $DMZ_IF -s $DMZ_NET              -p tcp --syn     --dport webcache

#
# reject auth on dmz interface
#

-A $R -j REJECT -i $DMZ_IF                       -p tcp --syn     --dport auth --reject-with
port-unreach

##
## FW-DMZ: allow connections from the fw to dmz net
##

R=FW-DMZ

-N $R

#-A $R -j LOG --log-prefix "$R "

#
# allow ping to dmz net
#

-A $R -j New -o $DMZ_IF          -d $DMZ_NET     -p icmp                  --icmp-
type ping

#
# allow smtp to dmz net
#

-A $R -j New -o $DMZ_IF          -d $DMZ_NET     -p tcp --syn     --dport smtp

#
# allow auth (when incomming smtp) to dmz net
#

-A $R -j New -o $DMZ_IF     -d $DMZ_NET          -p tcp --syn     --dport auth
```

```
#
# allow http to dmz net
#

-A $R -j New -o $DMZ_IF          -d $DMZ_NET        -p tcp --syn     --dport http

#
# allow ssh to dmz net
#

-A $R -j New -o $DMZ_IF          -d $DMZ_NET        -p tcp --syn     --dport ssh


45_fw-services-ext::

#
# services on the fw for extern
#

##
## Ext-FW: allow connections from extern to the fw
##

R=Ext-FW

-N $R

#-A $R -j LOG --log-prefix "$R "

#
# allow ping from extern
#

-A $R -j New -i $EXT_IF      -s $EVERYWHERE          -p icmp                --icmp-
type ping

#
# allow ssh for service from remote support addresses
#

-A $R -j New -i $EXT_IF      -s $EXT_SUPPORT1  -p tcp --syn     --dport ssh

-A $R -j New -i $EXT_IF      -s $EXT_SUPPORT2  -p tcp --syn     --dport ssh

#
# allow smtp from extern
#

-A $R -j New -i $EXT_IF -s $EVERYWHERE          -p tcp --syn     --dport smtp
```

```
#
# reject auth (when sending mail to extern mailserver)
#

-A $R -j REJECT -i $EXT_IF -s $EVERYWHERE    -p tcp --syn     --dport auth

#
# allow http from extern (go to twhttpd)
#

# -A $R -j New -i $EXT_IF -s $EVERYWHERE                -p tcp --syn     --dport http

#
# allow ike and esp from extern vpn-gateways (for ipsec)
#

#-A $R -j New -i $EXT_IF -s $VPN_GW1                -p udp  --sport ike --dport ike

#-A $R -j New -i $EXT_IF -s $VPN_GW1                -p esp

##
## FW-Ext: allow connections from the fw to extern
##

R=FW-Ext

-N $R

#-A $R -j LOG --log-prefix "$R "

#
# allow ping to extern
#

-A $R -j New -o $EXT_IF      -d $EVERYWHERE    -p icmp                --icmp-type ping

#
# allow domain-requests to extern (ISP) dns-servers
#

-A $R -j New -o $EXT_IF -d $EXT_DNS1    -p udp          --dport domain

-A $R -j New -o $EXT_IF -d $EXT_DNS2    -p udp          --dport domain

# also domain transfers

-A $R -j New -o $EXT_IF -d $EXT_DNS1    -p tcp          --dport domain

-A $R -j New -o $EXT_IF -d $EXT_DNS2    -p tcp          --dport domain
```

```
#
# allow ssh to manage extern machines
#

-A $R -j New -o $EXT_IF -d $EVERYWHERE        -p tcp --syn     --dport ssh

#
# allow smtp for sending out mail
#

-A $R -j New -o $EXT_IF -d $EVERYWHERE        -p tcp --syn     --dport smtp

#
# allow make auth (when getting mail from extern mailserver)
#

-A $R -j New -o $EXT_IF -d $EVERYWHERE        -p tcp --syn     --dport auth

#
# allow http/https for updating system and http-proxy
#

-A $R -j New -o $EXT_IF -d $EVERYWHERE        -p tcp --syn     --dport http

-A $R -j New -o $EXT_IF -d $EVERYWHERE        -p tcp --syn     --dport https

# allow connect proxy to special http-ports

-A $R -j New -o $EXT_IF -d $EVERYWHERE        -p tcp --syn     --dport 1024:65535 -m
owner --uid-owner proxy

#
# allow ftp for updating uvscan-dat
#

-A $R -j New -o $EXT_IF      -d $EVERYWHERE   -p tcp --syn     --dport ftp

#
# allow ike and esp to extern vpn-gateways (for ipsec)
#

#-A $R -j New -o $EXT_IF -d $VPN_GW1   -p udp  --sport ike --dport ike

#-A $R -j New -o $EXT_IF -d $VPN_GW1   -p esp

50_int-ext::

#
# direct routed/nated connections from intern to extern
#
```

```
##
## Int-Ext: allow connections from intern to extern
##

R=Int-Ext

-N $R

#-A $R -j LOG --log-prefix "$R "

#
# allow ping from intern net to extern
#

-A $R -j New -i $INT_IF       -o $EXT_IF -s $INT_NET     -p icmp                  --icmp-
type ping

#
# allow ssh from intern to extern
#

-A $R -j New -i $INT_IF -o $EXT_IF -s $INT_NET  -p tcp --syn     --dport ssh

#
# allow whois from intern to extern
#

-A $R -j New -i $INT_IF -o $EXT_IF -s $INT_NET  -p tcp --syn     --dport whois

#
# allow ftp from intern net to extern, ftp-proxy would be more secure
#

-A $R -j New -i $INT_IF -o $EXT_IF -s $INT_NET  -p tcp --syn     --dport ftp

#
# allow notes from intern to extern
#

-A $R -j New -i $INT_IF -o $EXT_IF -s $INT_NET  -p tcp --syn     --dport lotusnote

#
# allow vnc from intern to extern
#

-A $R -j New -i $INT_IF -o $EXT_IF -s $INT_NET  -p tcp --syn     --dport vnc1

-A $R -j New -i $INT_IF -o $EXT_IF -s $INT_NET  -p tcp --syn     --dport vnc2
```

```
#
# allow netbios from intern to ext server
#
-A $R -j New -i $INT_IF -o $EXT_IF -d 213.208.19.145 -p tcp --syn          --dport 137:139

-A $R -j New -i $INT_IF -o $EXT_IF -d 213.208.19.145 -p tcp --syn          --dport 445

#

53_int-dmz::

#
# direct routed connections from intern to dmz
#
##
## Int-DMZ: allow connections from intern to dmz
##

R=Int-DMZ

-N $R

#-A $R -j LOG --log-prefix "$R "

#
# allow ping from intern net to dmz
#

-A $R -j New -i $INT_IF        -o $DMZ_IF -s $INT_NET     -p icmp                  --icmp-
type ping

#
# allow ssh from intern to dmz
#

-A $R -j New -i $INT_IF -o $DMZ_IF -s $INT_NET -p tcp --syn      --dport ssh

#
# allow netbios from intern to dmz
#

-A $R -j New -i $INT_IF -o $DMZ_IF -s $INT_NET -p tcp --syn      --dport 137:139

-A $R -j New -i $INT_IF -o $DMZ_IF -s $INT_NET -p tcp --syn      --dport 445

#
# allow ftp from intern net to dmz
#

-A $R -j New -i $INT_IF -o $DMZ_IF -s $INT_NET -p tcp --syn      --dport ftp
```

```
#
# allow notes from intern to dmz
#

-A $R -j New -i $INT_IF -o $DMZ_IF -s $INT_NET -p tcp --syn      --dport lotusnote

#
# allow vnc from intern to dmz
#

-A $R -j New -i $INT_IF -o $DMZ_IF -s $INT_NET -p tcp --syn      --dport vnc1

-A $R -j New -i $INT_IF -o $DMZ_IF -s $INT_NET -p tcp --syn      --dport vnc2


55_dmz-ext::

#
# direct routed/nated connections from dmz to extern
#
##
## DMZ-Ext: allow connections from dmz net to extern
##

R=DMZ-Ext

-N $R

#-A $R -j LOG --log-prefix "$R "

#
# allow ping from dmz net to extern
#

-A $R -j New -i $DMZ_IF      -o $EXT_IF -s $DMZ_NET    -p icmp                  --icmp-
type ping

#
# allow ftp from dmz net to extern, ftp-proxy would be more secure
#

-A $R -j New -i $DMZ_IF -o $EXT_IF -s $DMZ_NET        -p tcp --syn      --dport ftp

#
# allow direkt smtp from dmz to extern
#

-A $R -j New -i $DMZ_IF -o $EXT_IF -s $DMZ_NET        -p tcp --syn      --dport smtp
```

```
#
# allow notes from dmz to extern
#
-A $R -j New -i $DMZ_IF -o $EXT_IF -s $DMZ_NET        -p tcp --syn     --dport lotusnote

#

60_ext-dmz::

#
# direct routed/nated connections from extern to dmz
#
##
## Ext-DMZ: allow connections from extern to the dmz net
##

R=Ext-DMZ

-N $R

#-A $R -j LOG --log-prefix "$R "

#
# allow notes from extern to dmz
#

-A $R -j New -i $EXT_IF -o $DMZ_IF -s $EVERYWHERE  -p tcp --syn     --dport lotusnote

#
# allow pop3 from extern to dmz
#

-A $R -j New -i $EXT_IF -o $DMZ_IF -s $EVERYWHERE  -p tcp --syn     --dport pop3

#
# allow vnc from extern to dmz
#

-A $R -j New -i $EXT_IF -o $DMZ_IF -s $EVERYWHERE  -p tcp --syn     --dport vnc1

-A $R -j New -i $EXT_IF -o $DMZ_IF -s $EVERYWHERE  -p tcp --syn     --dport vnc2

#
# allow ftp from extern to dmz
#

-A $R -j New -i $EXT_IF -o $DMZ_IF -s $EVERYWHERE  -p tcp --syn     --dport ftp
```

```
#
#

90_nat::

#
# gerneral rules for SNAT, DNAT, Masquerading
#
#
# SNAT, Masquerading
#
# Masquerade packets going out the extern interface
# good for dynamic ext address

# -t nat -A POSTROUTING -o $EXT_IF       -j MASQUERADE

# Alternative SNAT
## Change source addresses to $EXT_IP

-t nat -A POSTROUTING -o $EXT_IF         -j SNAT         --to $EXT_IP

## Change source addresses to mulitple $EXT_SNAT_IP1 - $EXT_SNAT_IP2

# -t nat -A POSTROUTING -o $EXT_IF       -j SNAT         --to $EXT_SNAT_IP1-
$EXT_SNAT_IP2

## Change source addresses to $EXT_IP, ports 1-1023

# -t nat -A POSTROUTING -p tcp -o $EXT_IF        -j SNAT --to $EXT_IP:1-1023

#
# DNAT
#

## Change destination addresses to 5.6.7.8

# -t nat -A PREROUTING -i $INT_IF          -j DNAT --to 5.6.7.8

## Change destination addresses to 5.6.7.8, 5.6.7.9 or 5.6.7.10.

# -t nat -A PREROUTING -i $INT_IF          -j DNAT --to 5.6.7.8-5.6.7.10


## Change destination addresses of web traffic to 5.6.7.8, port 8080.

#-t nat -A PREROUTING -i $INT_IF -p tcp --dport 80 -j DNAT --to 5.6.7.8:8080


## Change destination addresses of vnc,pop3,notes traffic to dmz

#-t nat -A PREROUTING -i $EXT_IF -p tcp --dport vnc1 -j DNAT --to 172.16.0.10
```

```
#-t nat -A PREROUTING -i $EXT_IF -p tcp --dport vnc2 -j DNAT --to 172.16.0.10

#-t nat -A PREROUTING -i $EXT_IF -p tcp --dport pop3 -j DNAT --to 172.16.0.10

#-t nat -A PREROUTING -i $EXT_IF -p tcp --dport lotusnote -j DNAT --to 172.16.0.1

#-t nat -A PREROUTING -i $EXT_IF -p tcp --dport ftp -j DNAT --to 172.16.0.1


## Send incoming port-80 web traffic to our squid (transparent) proxy

#-t nat -A PREROUTING -i $INT_IF -p tcp --dport 80 -j REDIRECT --to-port 8080

#-t nat -A PREROUTING -i $DMZ_IF -p tcp --dport 80 -j REDIRECT --to-port 8080


#

95_vpn::

#

# direct routed connections on vpn

#
##
## Int-VPN: allow connections from intern to vpn
##

#R=Int-VPN

#-N $R

#-A $R -j LOG --log-prefix "$R "

#
# allow all from intern net to vpn
#

#-A $R -j New -o $VPN_IF -s $INT_NET      -p all

#
# TMP allow all on vpn
#

-I OUTPUT -j ACCEPT -o ipsec0

-I INPUT -j ACCEPT -i ipsec0

-I FORWARD -j ACCEPT -i ipsec0
```

-I FORWARD -j ACCEPT -o ipsec0

#

99_main::

##
main rules
##

 # packets into the fw

-A INPUT -j Junk

-A INPUT -j FW-FW_INPUT

-A INPUT -j Established

-A INPUT -j Int-FW

-A INPUT -j DMZ-FW

-A INPUT -j Ext-FW

-A INPUT -j Drop

 # packets out of the fw

-A OUTPUT -j FW-FW_OUTPUT

-A OUTPUT -j Established

-A OUTPUT -j FW-Int

-A OUTPUT -j FW-DMZ

-A OUTPUT -j FW-Ext

-A OUTPUT -j Drop

 # packets through the fw

-A FORWARD -j Junk

-A FORWARD -j Established

-A FORWARD -j Int-DMZ

-A FORWARD -j Int-Ext

-A FORWARD -j DMZ-Ext

-A FORWARD -j Ext-DMZ

-A FORWARD -j Drop

 # enable rules

--policy INPUT DROP

--policy FORWARD DROP

--policy OUTPUT DROP

#

4.4 Flow of the Program

Functions and Program Flow

- ✓ Check if it is TCP/IP packet.
- ✓ Get packet info and flow info.
- ✓ Process_found() -> create_if()
- ✓ process()
- ✓ finish()
- ✓ insert()
- ✓ process_drop()

The various functions in the module and the flow of execution in the program are as follows. For each incoming packet we first check if it is a TCP/IP packet, this is done by checking if (sb->nh.iph->protocol) == 6. If the packet is a TCP/IP packet, then we get information about the packet as well as other data that gives us information about the flow to which the packet belongs. This information consists of the source IP Address, Destination IP Address, Source Port, Destination Port, TCP Header Length and the status of Fin Flag in the packet.

Then the function process_found() is called. In this process we traverse the entire TCP data until we reach the end of TCP data marked by (sb->tail). During the traversal if the search

string is found, then we set the 'found' flag to 1 and break from the loop to avoid further traversal of TCP Data. If even at the end of TCP Data the pattern is not found then we do not change the value of 'found' flag which by default is set to 0. By looking at the found flag at any future point of packet processing we can tell if it contains the search string or not.
After process_found() has done it's processing we call the function finish() else we call the function process(). In the functions finish() and process() we do certain processing required for maintaining the status of the flows, also if the 'found' flag is set to 1 we increment the value of count for the flow in the node by one else if 'found' flag is set to 0 we do not make any change to count. If during processing of functions finish() and process() we find that the packet is the first packet of the flow, we create a new node for the flow by calling the function insert(). In the function insert() we create a new node of the type sock_detail and enter important information like source IP Address, Destination IP Address, Source Port, Destination Port. Also we set the proper values of limit, count, finish. Then we insert the node at the end of the linked list.
Then if 'found' == 1 we call the function process_drop(). In process_drop(), we check if the count for this flow has exceeded the limit of the flow, this is done by checking if (count > limit). If (count > limit), then 'drop' flag is set to 0 else 'drop' flag is set to 1. Finally if 'drop' flag is set to 1 the packet is dropped using NF_DROP return type, else if 'drop' flag is set to 0 the packet is let through using NF_ACCEPT return type.

4.5 Function Description:

process_found():

This function traverses the entire TCP Data and simultaneously checks if the search string is found. If the search string is found the traversing of TCP Data is stopped and the 'found' flag is set to 1, before the function process_found() ends it's processing. If the search string is not found at the end of the traversal then the function process_found() ends it's processing without changing the value of 'found' flag which by default is set to 0. In this project a naïve pattern matching algorithm has been used. This can be improved in future.

process():

This function checks if the flow which the packet belongs to exists in the linked list, if it doesn't this function calls the function insert() else it finds the relevant node and does further processing. Further it checks if the 'found' flag is set to 1. If 'found' flag is set to 1, the value of count is incremented by 1 else the value of count is not changed. Subsequently if the Fin flag has already been encountered for this flow, it implies that this packet is the "ack" packet for the Fin packet, hence we delete the node from the link list and make the linked list proper.

insert():

Thisfunction is called when the router encounters the first packet of any flow. In this function we create a new node of the type sock_details and insert the various values representing the flow into this node. The various values include source IP Address, Destination IP Address, Source Port, Destination Port. Also we set the proper values of limit,

count, finish. Then we insert the node at the end of the linked list, which is pointed to by head_list.

finish():

This function is called if theFin flag in the packet is set. In finish() function firstly we traverse the entire linked list of nodes until we get the node representing the flow of the packet or we reach the end of the list. If we find the flow in the linked list then we set the finish flag of the flow to 1 indicating that for this flow the Fin flag has been encountered. If we reach the end of the list and still do not find the node that indicates that it is the first packet of the flow to be encountered by the router, hence we create a new node using the function insert() and append it to the existing linked list. Finally we set the finish flag to 1 for this newly created node.

process_drop():

This functionis called only if 'found' flag is set to 1, i.e. if the search string is found in the TCP Data. In this function we check if we have check if the limit has been reached for the flow that the current packet belongs to. This is done by checking if (count > limit). limit indicates the number of packets containing the search string that need to be dropped for each flow. Hence if (count > limit), it indicates that for that flow, we have already dropped the required number of packets and any subsequent packets of that flow containing the search string shouldn't be dropped. So if (count > limit) 'drop' flag is set to 0 else drop 'flag' is set to 1 indicating that this packet needs to be filtered i.e. dropped.

4.6 Useful Calculations:

In this project we are working at the IP Layer and we have to access the data of TCP layer. Due to this layer mismatch, in order to prevent a layering violation, the kernel does not let us use the direct pointer to the TCP part using skb->h.th. In order to access TCP part of the sk_buff, we have to use the available pointer in IP layer and access the TCP Part by doing Pointer Arithmetic.
The structure of IP and TCP Header is as follows.

We can reach the start of the IP Header by using the available pointer sb->data. Further we can get the length of the IP header using the pointer (sb->nh.iph->ihl). Now in order to reach the start of TCP Header we have to take a pointer that points to the start of IP header and move it forward by the length of IP Header. To achieve this we use

v = (sb->data) + (sb->nh.iph->ihl);

where (sb->data)denotes the start of IP Header

and (sb->nh.iph->ihl)denotes the length of IP Header.

Further in order to access the TCP Data we have to access a pointer that point to the start of TCP Header (i.e. v) and increment this pointer by the length of TCP Header. To achieve this we use

v1 = (v + (TCP_HLEN *4));

where v = (sb->data) + (sb->nh.iph->ihl); denotes the start of TCP Header
and TCP_HLEN = ((*(v + 12)) / 16); denotes the length of TCP Header because (v+12) points to the TCP Header length and hence *(v + 12) denotes the value of TCP Header length.

CHAPTER 5
TESTING

5.1 TESTING

Testing is the process of executing the program(s) with the intention of finding out errors. During testing, the program to be tested is executed with a set of test cases and the output of the programs for the test case is evaluated to determine if the program is performing as it is expected to be. The success of testing in revealing errors in programs depends critically on the test cases

5.1.1 LEVELS OF TESTING

UNIT TESTING: The first level of testing is called unit testing. In this different modules are tested against the specifications produced during design of the modules. Unit testing is essentially for verification of the code produced during coding phase, and hence the goal is to test the internal logic of the modules. The programmer of the module typically does it.

INTEGRATION TESTING: The next level of testing is often called integration testing. In this, many unit-tested modules are combined into subsystems, which are then tested. The goal is here to see if the modules can be integrated properly. Hence, the emphasis is on testing interfaces between modules. The testing activity can be considered testing the design. The integration plan specifies the steps and order in which modules are combined to realize the full system. After each integration step, the partially integrated system is tested. An important factor that guides the integration is the module dependency graph.

SYSTEM TESTING: System tests are designed to validate a fully developed system to assure that it meets its requirements. There are essentially three main kinds of system testing:

ALPHA TESTING: Alpha refers to the system testing carried out by the test team within the developing organization.

BETA TESTING: Beta testing is the system testing performed by a select group of friendly customers.

ACCEPTANCE TESTING: Acceptance testing is the system testing performed by the customer to determine whether to accept or reject the delivery of the system.

5.1.2 TYPES OF TESTING:

BLACK BOX TESTING: This testing is also known as functional testing. The basis for deciding test cases in functional testing is the requirements or specifications of the system or modules. For the entire system test cases are designed form the requirement specification document from the system. There are number of techniques that can be used to select test cases that have been found to be very successful in detecting errors.Some of them are:

Equivalence class partitioning: In this we divide the domain of all the inputs into a set of equivalence classes. That is we want to identify classes of test cases such that the success of one test case in a class implies the success of others. It is often useful to consider equivalence classes in the output. For an output equivalence class, the goal is to generate test cases such that the output of that test case lies in the output equivalence class.

Boundary value analysis: In boundary value analysis, we choose an input for a test case from an equivalence class, such that the input lies at the edge of the equivalence class. Boundary value rest cases are also called "extreme cases".

Cause-effect graphing: It is a technique that aids in selecting combinations of input conditions in a systematic way. A cause is a distinct input condition, and an effect is n distinct output condition. Each condition forms a node in the cause-effect graph. Beyond

generating high-yield test cases, it also aids the understanding of the functionality of the system, because the tester must identify the distinct causes and effects.

Special cases: It depends on the data structures and the function of the module. There are no rules to determine special cases, and the tester has to use his intuition and experience to identify such test cases. Consequently, determining special cases is also called "error guessing.

WHITE BOX TESTING: There are several white box-testing strategies. Each testing strategy is based on some heuristic. One white box testing strategy is said to be stronger than another strategy, if all types of errors detected by the first testing strategy (say B) are also detected by the second testing strategy (say A), and the second strategy additionally detects some more types of errors. When two testing strategies detect errors that are different at least with respect to some types of errors, they are then called complementary.

Statement coverage: It aims to design test cases so that every statement in the program is executed at least once. The principal idea is that unless we execute a statement, we have no way of determining if error exists in that statement.

Branch coverage: In this strategy, test cases are designed to make each branch condition assume true and false value in turn. It is also known as "edge-testing" as in this testing scheme, each edge of a program's control flow graph is traversed at least once.

Condition coverage: In this test cases are designed to make each component of a composite conditional expression assume both true and false values. Thus, condition testing is a stronger testing strategy than branch testing. For conditional coverage, the number of test cases increases exponentially with the number of component conditions.

Path coverage: It requires us to design test cases such that all linearly independent paths in the program are executed at least once. These paths are defined in terms of control-flow graph of a program.

As testing forms the first step towards determining the errors in a program it should be properly carried out.

UNIT TESTING:

Unit testing was carried out for each module against the specifications produced during the design of the module.

Unit testing of each program and module was done with the following perception.

USER INTERFACE:

User interface was tested which gave rise to more user understandable errors and help messages.

INTERNAL LOGIC:

While testing a module, the internal logic was tested.

INTEGRATED TESTING:

Tint the integrated testing, the unit-tested modules were combined into subsystems and then tested.

The strategies for integrated tested comprised of :

- Performance time testing.
- Logical cycle of data.
- Test data.
- Live data obtained from users.

5.1.3 Firewall Testing:

Firewall testing is very important, it gives confidence to the administrator that the firewall rules they write are working properly. The right packet is accepted and right packet is dropped.

Firewall testing is difficult because there are many parameters which resulting is huge number of possible parameter combination.

There are number of possible combination of test cases can be used to test firewall rules. Typically each test case is viewed as a row in a table or in database term, a relation. So the main problems are:

1. Test case selection: which test cases should be applied? Different mathematical calculation can be used for it.

2. Test case execution: After selecting test cases, what method should be applied to execute it?

For first problem an efficient algorithm will be sufficient. For second there are libraries to factor out the many details of packet generation, transmission and reception .

Iptables are used for packet filtering based on header fields e.g., IP address, TCP and UDP port and TCP flag. Four main features of iptables are stateless filtering, state full filtering, network address/port translation, and logging.

The syntax of iptables is simple. Typically rule is ACCEPT or DROP. The rules work like C switch statement. If Pi matches, then ai is invoked. If no predicate matches, default action is make (ACCEPT or DROP). Usually default DROP is chosen for security reason. Iptables are implemented using Linux command line. Testing configuration consist of two PCs- the driver and system under test (SUT)- The SUT configured so that traffic enter from eth1 with destination IP address and routed to eth2 and vice versa. ARP (Address Resolution Protocol) packets are used to map IP (Internet Protocol) address to MAC (Medium Access Control) addresses.

The first issue is how to generate, send, and receive frames on the drivers eth1 and eth2 interface. They have developed a raw socket library which makes the resulting test cases much easier to understand and modify. So at the end result is a simple open/close/read/write interface, much like the Linux raw I/O interface.

They created test template by providing different combination of parameters. There are three strategies they used for Tuple generation: Cartesian product generation, boundary value generation and pair wise generation. The testing framework for iptables has been implement in a tool called PBit (Pattern Based iptables tester). The useful feature of PBit is that you can modify the test configuration at run time.

5.1.3.1 Test Case Table:

Test Case ID	Activity	Status	Expected Output
1	Login to FTP	INVALID	No Login Screen
2	Login to http	VALID	Redirect to the Page
3	Connecting to 172.16.28.241	INVALID	Not Allowed/Blocked
4	Connecting to 172.16.28.41	VALID	No error.
5	Broadcasting	INVALID	Not Allowed
6	Ping	VALID	Allowed
7	ssh	VALID	Allowed
8	netbios	VALID	Allowed
9	Use 8080 port	INVALID	Blocked
10	Connecting to 172.16.73.12	VALID	Allowed
11	Connecting to 172.16.73.3	VALID	Allowed

Table 8. Test Case

5.1.3.2 Practical Test:

In this section we will describe the testing method implementation, like Programming language or a scripting language that is used to implement an automated test method

The test was performed using the nmap tool by executing manually a set of nmap scans and collecting results. The types of scans performed were:

TCP scan

SYN scan

FIN scan

UDP scan

IP protocol scan

ACK Scan

List Scan

Version Detection

Each scan command was executed for each network address space, the DMZ address space and the LAN address space.

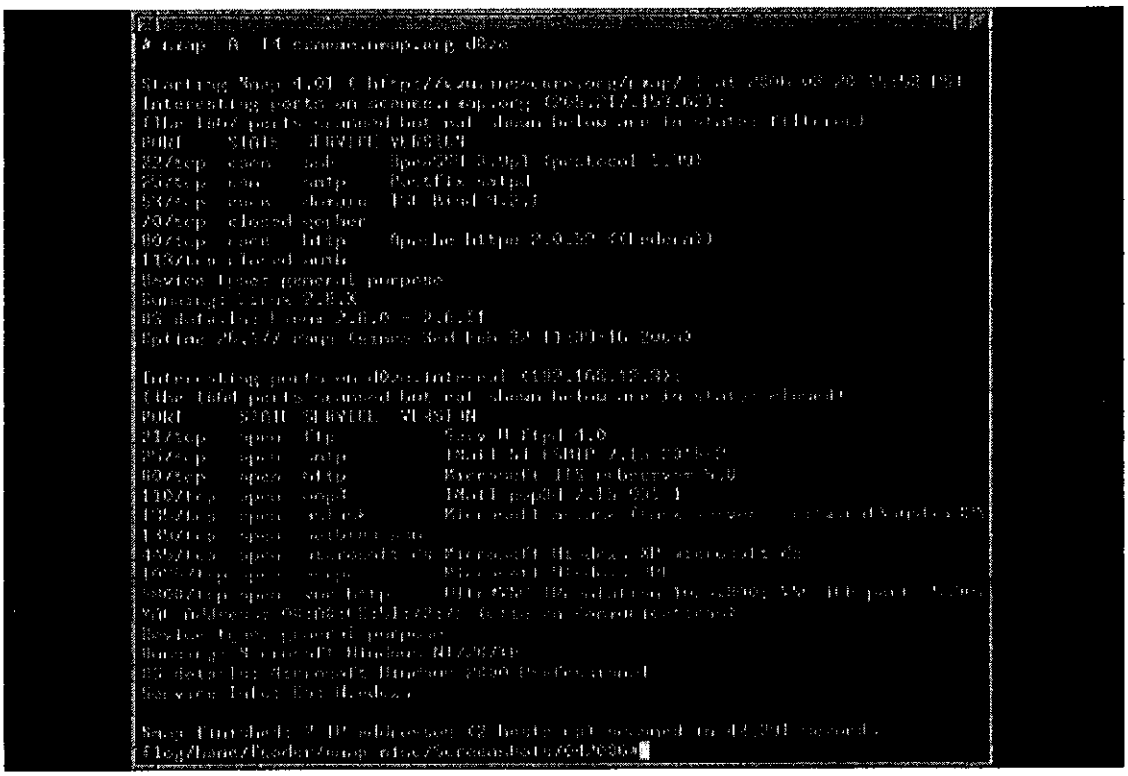

Figure 12. nmap

After performing a testing using nmap tool and using it to scan and probe the other group firewall we found the following:

1- The firewall Mac Address which

is FE:FD:00:00:ED 6.

2- the open port, the protocol and the service of the

dmz.web server:

PORT STATE SERVICE

80/tcp open http

3- The number of the up hosts on the dmz network, which are two hosts.

4- The available open services on the firewall external interface eth0, Lan interface eth1 and DMZ interface eth2 which are:

PROTOCOL STATE SERVICE

1 open icmp

5- Identified unfiltered ports on the dmz.web server

which are:

PORT STATE SERVICE

80/tcp unfiltered http

5.1.3.3 Technologies:

In this section we will describe the technologies that are using in the project environment like UML "User Mode Linux", MLN tool and iptables

So we will start with the "UML". UML is a port of Linux to the Linux system call interface, and allows users to run any number of virtual systems (UML instances) without the need for special privileges. The "UML" system also includes basic facilities for networking virtual machines. [4]

To simplify setup of networks of UML instances, a tool called "MLN" is use. MLN (Manage Large Networks) is a virtual machine administration tool designed to build and run virtual machine networks based on Xen, VMWare Server and User-Mode Linux. It is ideal for creating virtual network labs for education, testing, hosting or simply playing around with virtual machines. The goal is to ease the configuration and management of virtual networks. Xen and User-Mode Linux are widely used as tools for testing, learning and virtual hosting. MLN builds and configures file system templates based on its descriptive and easy programming language and stores them in an organized manner. It also generates start and

stop scripts for each virtual host, enabling you to manage a running virtual network by stopping individual virtual machines within a network and starting them again. MLN makes it possible to have several separate networks, projects, at once and even connect them together to create larger networks. [5]

Ipchains are the most common firewall/Nat packages running on linux. Iptables are the enhanced product of ipchains by net filter organization.

Iptables is packet filtering firewall software. Packets inspected by iptables are passing through sequence of rules.

There are total three types of tables.

Â¢ Mangle table

Â¢ Filter queue

Â¢ Nat queue

Mangle table is responsible for alteration of quality of service bit in TCP header. This is very rarely used in satellite offices and home offices (SOHO) environment. Filter queue is responsible for packet filtering. It contains three built-in chains where we can put rules. These are:

Forward chain, Input chain and output chain. Nat queue contains two built-in chains. Which are pre-routing and is responsible for Network address translation. It post-routing?

Queue Type Queue Function Packet

Transformation Chain in Queue Chain Function

Mangle TCP header modificatio n PREROUTING

POSTROUTING

OUTPUT

INPUT

FORWARD Modification of the TCP packet quality of service bits before routing occurs. (Rarely used in SOHO environments)

Filter Packet filtering FORWARD Filters packets to servers accessible by another NIC on the firewall.

INPUT Filters packets destined to the firewall.

OUTPUT Filters packets originating from the firewall

Nat Network Address Translation PREROUTING Address translation occurs before routing. Facilitates the transformation of the destination IP address to be compatible with the firewall's routing table. Used with NAT of the destination IP address, also known as destination NAT or DNAT.

POSTROUTING Address translation occurs after routing. This implies that there was no need to modify the destination IP address of the packet as in pre-routing. Used with NAT of the source IP address using either one-to-one or many-to-one NAT. This is known as source NAT, or SNAT.
OUTPUT Networks address translation for packets generated by the firewall. (Rarely used in SOHO environments)

For every firewall rule, user needs to specify iptables and ipchain. As most of the rules are related to filtering, so if user mentions any rule without an associated table then it will be considered as a part of the filter table. So we can say filter table is a default table.
In the figure 2 [2] packets arrives from Network and handled by the firewall to create a data connection.
The packet is first examined by the mangle table PREROUTING chain, then it is pass through nat table PREROUTING chain to check, it DNAT of not. Then its sent for routing. If it is defined to protected network then it passes through. FORWARD chain of mangle table for quality purpose then it is filtered by the rules of the filter table in its forward chain and if necessary it goes for
SNAT in POSTROUTING chain of mangle table and
then it arrives to network B. If the destination wants to apply, it will follow the same sequence.
If packet is passed through the firewall then it through INPUT chain of the mangle table, then it is filtered by a INPUT chain of the filter table. Then it passes to the firewall application for some processing.
If firewall reply then packet is sent for the routing and it is inspected by OUTPUT chain of the mangle table, if any.
Then OUTPUT chain of NAT table see if any DNAT
is required then OUTPUT chain rule of filter table are applied to that packet.
Finally POSTROUTING chain of mangle checks the QoS of packet and then POSTROUTING chain of nat
table check SNAT of the packet.

5.1.3.4 Environments

The environment on which we have performed the firewall configuration and testing was a linux workstation that has virtual network of virtual computers that are managed by the

MLN tool. The network schema is described in the lab document. The tests are done using external and internal UML instances.

CHAPTER 6
CONCLUSION

Conclusion:

The rules in this firewall can prevent following situations:

- ✓ Spoofing
- ✓ Stateful Checking
- ✓ Nat rules
- ✓ Vpn rules
- ✓ Dmz rules

Future Recommenations:

- ✓ Ability to search a pattern or regular expression rather than a strict string. Eg.xx.xx.xx.xx
- ✓ To do flow maintenance based on time i.e. using **"jiffies"** to insert timestamps to the nodes whenever they are accessed. In this method the dead flows will not be accessed for a longer time as compared to the flows alive, hence we can delete all the dead flows, which have older timestamps.
- ✓ To allow different keywords for different flows.
- ✓ Implementing functionality to allow user input of the search keyword.
- ✓ To use KVM Algorithm or Suffix trees to pre-process the string in order to improve the search time.

BIBLIOGRAPHY

- Behrouz A. Forouzan, 2003, TCP/IP Protocol Suite, Mc Graw Hill.
- A Map of the Networking Code in Linux Kernel 2.4.20 by M. Rio et al. 31 March 2004.
- Hacking the Linux Kernel Network Stack http://www.phrack.org/show.php?p=61&a=13
- The "Networking" code in Linux, Teunis J. Ott and Rahul Jain July 29, 2004
- D0 Code – comprehensively cross – referenced and searchable code http://www-d0.fnal.gov/D0Code/source/
- The journey of a packet through the linux 2.4 network stack - Harald Welte http://gnumonks.org/ftp/pub/doc/packet-journey-2.4.html
- The netfilter framework in Linux 2.4 - Harald Welt
- Interactions http://linux-ip.net/html/adv-overview.html
- http://gnumonks.org/papers/netfilter lk2000/presentation.html Overview of Routing and Packet Filter
- The Packet handling in default Linux kernel 2.4 has been taken from http://open-source.arkoon.net/kernel/kernel_net.png