

# Chapter 11

## Router Configuration

This chapter introduces how a Linux system can function as a Router. We will create a simple dual Ethernet system, but the concepts can be applied to systems that support additional interfaces.

### Concepts Learned in this Chapter

- Setting up multiple Ethernet interfaces
- Networking Routing
- Network Address Translation
- Firewall Design

## Table of Contents

Router Configuration.....	1
11.1 Dual Ethernet.....	3
11.1.1 IP Address.....	4
11.1.2 Gateway Address.....	4
11.2 Dual Ethernet Router.....	4
11.2.1 Equipment Requirements.....	5
11.2.2 Objective 1: Installation of Two NIC cards.....	5
11.3 Route Command.....	7
11.3.1 Examples of usage.....	9
11.4 Network Address Translation Router (NAT).....	9
11.4.1 IP Masquerading, .....	10
11.4.2 IP Forwarding (Not Complete).....	11
11.5 IP Chains Firewall,,.....	12
11.5.1 Firewall Background.....	12
11.5.1.1 Action.....	13
11.5.1.2 Chain.....	13
11.5.1.3 Setting up a Filtering Firewall.....	15
11.5.2 A Working Filter.....	17
11.6 IP Tables Firewall .....	26
11.6.1 Using IPTables.....	27
11.X Commands Used in this Chapter.....	30
11.Y Chapter Review Questions.....	30

## 11.1 Dual Ethernet

Most commonly we address the installation of Ethernet NICs, but the same concept will also apply to any other network interface, including Token Ring and a serial interface. In this practice, we address only the Ethernet interface. The following information is derived from the **Ethernet HOWTO**.

Fortunately, with the present distributions, nearly all Ethernet cards are automatically detected by the **modprobe** utility to assign the appropriate driver, assigning the correct hardware address and interrupt values. Whether they are installed prior to installation or afterwards, upon the booting of Linux after the installation, they will be detected and allow configuration.

If additional options are required, then one must configure the Unix and Linux distributions to use modular drivers, that is, external drivers that are not built into the kernel. If the card is of the PCI (Plug and Play) type, the module will typically detect all of the installed cards. For ISA cards, we typically need to specify the Interrupt and I/O base address for each card. For the boot process, this information may be stored in the **/etc/conf.modules** file.

The **conf.modules** format is:

```
alias      eth0      type1
alias      eth1      type2
options    -o type1   io=0xabc   irq=m
options    -o type2   io=0xdef   irq=n
```

Experience has shown that although you can install two different vendors, fewer problems will be encountered if the same vendor is used for all cards. When you have a Pentium PC, using PnP cards generally makes this problem a non-issue.

The above configuration of PnP cards allows the **modprobe** to assign the appropriate driver and to assign the correct address and interrupt values. Note that for non-PnP cards, the I/O address and interrupt values are those configured on the card in the EEPROM or by switches or jumpers on the card. In order to change the EEPROM values, one must use the vendor's configuration program, which normally operates under DOS.

After the cards have been booted up, they can be tested with the commands:

```
modprobe ethN
dmesg | less
```

where N is the interface number.

At bootup, Linux may only probe for the first available card, leaving the second (or more) cards uninstalled. We can manually force the probing of the second card at the LILO command by giving the command:

```
LILO: linux ether=5, 0xadd1, eth0 ether15, 0xadd2, eth1
```

This can be cumbersome to enter every time at boot, so we can make it automatic by using the **append** command in the **/etc/lilo.conf** file.

During the installation of Linux, the process will normally auto detect both (or all) cards on a newer version. You can configure the IP address of the first card during installation, but will be required to configure the address of the other card(s) later.

Setting up the second or later card is most easily accomplished using the X Windows vendor supplied Network Configurator. Using either KDE or GNOME, you are able to set the various addresses and subnet masks for each interface.

Note that when configuring an interface, at least three settings must be made to each interface – IP Address, Subnet Mask, and Gateway. The DNS address may also be required, but these are not directly related to the interface.

### **11.1.1 IP Address**

For a review of IP address operation, go back to Chapter 8 to obtain more detail of how addresses are utilized. Remember that a host address consists of both the IP address and the subnet mask.

### **11.1.2 Gateway Address**

Finally, a host needs to know where a packet must be sent if it is not destined for the local network. This is known as the Gateway Address. Typically, it is applied to each host, and not to a router, but sometimes we need to specify to the router which other device to utilize in order to forward packets. This device, a router, will have a local network address.

## **11.2 Dual Ethernet Router**

A router provides for the interconnection of different LAN segments or domains. It is capable, if necessary, of converting the protocol on one LAN segment to another format for the transmission over extended distances. Normally, a router is intended to maintain the same protocol on both segments, whereas a Gateway is capable of converting protocols if such is required. Because of marketing's improper use of terms, the use of router and gateway are often used interchangeably.

Routers are able to support the following functions:

1. Efficiently direct packets from one network to another, thereby reducing network traffic.
2. Join close or far networks.
3. Connection of different network protocols.
4. Reduction of network segment bottlenecks.
5. Function as a local network segment firewall

The setting up of a router requires several steps. First we must set up a computer by using the Linux Operating System to support multiple routing interfaces. Second, we then configure these interfaces as a routed network. This should have been done in the previous lab.

Recalling from previous discussion that in general our communications software is built upon a model called the OSI 7 Layer Model. Within this model, the forth layer, Networking, is where the routing function takes place. Instructing the router to know how to route to different networks can be achieved in two different ways. The first is to manually enter the values, called **Static Routing**, and the second method is for the router on it own to automatically learn who is “out there”, called **Dynamic Routing**.

Static Routing is where the system administrator must manually enter the routes to a routing table via various commands either at the Command Line Interface or through X Windows GUI interfaces.

Dynamic Routing, when enabled, allows the router to listen to the network and learn what other segments exist, and thereby automatically add that network segment address to its route table. This is the preferred method because the administrator is not required to be notified and manually update the table – which would be a considerable problem to their normal workload. During Dynamic Routing, packets are exchanged between routers that contain “routing table entries”, which are used to update the local table entries.

Red Hat Linux version 6.0 and later may be easily configured to support all of the routing and firewall functions. Although Linux is capable of supporting multiple protocols, we will concentrate on the TCP/IP protocol for this configuration. Other protocols will require the user to do independent investigation.

### 11.2.1 Equipment Requirements

The following equipment is required to set up a router:

1. 486 PC or better
2. 32 MB RAM or greater
3. 500 MB Hard Drive, 1.5 GB preferred
4. 2 Ethernet NIC cards
5. CD ROM
6. Red Hat Linux version 6.0 or later

These are the minimum requirements, especially with the Hard Drive, and require that a minimum installation of Linux be performed. An improved system, specifically a Pentium with 128 MB of RAM and a 6 Gbyte drive is recommended for installing more up to date distributions. This type of installation is a minimum system, working in the Command Line Mode only.

### 11.2.2 Objective 1: Installation of Two NIC cards

Our first requirement will be to install two NIC cards in the computer. It is assumed that at least a minimum installation has been made and that the system is operating correctly. If the system was not booted to X Windows automatically, then start it. It is also assumed that the two NIC cards were previously installed in the computer.

Although the write up specifies IP addresses, you need to be aware that you may need to use other addresses either in the lab configuration or in a real network.

1. Log onto the system as the root administrator.
2. Right Click on the desktop (not on an icon), then left click on **Execute Command**. Type in **netcfg** (this command unfortunately is depreciated) and hit enter to start the application. Newer versions should use the X Windows Network Configurator.
3. The Network Configurator will appear. Click on the Interfaces Tab.
4. If Linux saw the either or both of the cards, you should see **eth0** and / or **eth1**, if there is something different, delete everything but the **lo** entry.
5. Assuming that neither Ethernet interface card was observed, click on **ADD** and select **Ethernet**, click on **OK**.
6. We need to set the IP address for **eth0** to **198.168.0.100**. (*This address needs to be adjusted appropriately for your specific LAN segment. See below.*)
7. Click **ACTIVATE INTERFACE AT BOOT TIME** and click **DONE**.
8. Click on **SAVE** configuration to implement the update for the first Ethernet interface.
9. Select the installed device (eth0), click on **ACTIVATE**.
10. We now need to set up the second interface (**eth1**). Repeat steps 5, 6, 7, 8, and 9 for the second Ethernet card, with the exception that we need to utilize the IP address **192.168.10.100**. (See below.)
11. Save the changes and exit.
12. If the configuration is not current. Investigate what needs to be done and activate the changes.
13. As in Step 2, again open the **linuxconf** application from the desktop.
14. Click on the **Networking Tab**, then click on the **Basic Host Information** button. Finally click on **Adaptor 1**.
15. Make sure that the interface is enabled, has the Primary Name of **Router10**, with the domain name of **alab.com**. There should be no **aliases**. Make sure the IP address is **192.168.10.100** and that the subnet mask is **255.255.255.0** (or as specified by the instructor). In the kernel module drop down box, select the type of NIC card that was installed. Click the accept **Button**.
16. Now select **Adaptor 2**. Make sure that it has the Primary Name of **Router20**, with the domain of **blab.com**. There should be no **aliases**. Again confirm that the IP address is **192.168.0.100**. and that the subnet mask is **255.255.255.0**. In the Kernel module drop down box, select the type of NIC card that was installed. Click the accept **Button**.
17. Exit everything – returning back to the desktop.
18. Reboot the computer.

You should now have a system with two working NIC cards that have been activated during the boot. If you do another reboot, you should be able to observe the lines during the boot process “**bringing up interface eth0 and**

**eth1**". Issue the command **ifconfig** to confirm that both ports are active and have the proper IP addresses. It is a good idea to provide some physical marking on the computer to indicate which physical interface card is which. During the documentation, note the IRQ and IO addresses of each interface. This may take some experimentation to confirm which is which.

For a discussion, the following addresses will be used:

Segment 1: 192.168.0.0 Host: 192.168.0.2 Rtr 1: 192.168.0.100	Segment 2: 192.168.10.0 Host: 192.168.10.2 Rtr 1: 192.168.10.101 Rtr 2: 192.168.10.101
Segment 3: 192.168.20.0 Host: 192.168.20.2 Rtr 2: 192.168.20.100 Rtr 3: 192.168.20.101	Segment 4: 192.168.30.0 Host: 192.168.30.2 Rtr 3: 192.168.30.100

### 11.3 Route Command

Before we can have a computer communicate with the Internet, or anyone else on our local network, it must know what information is to be sent where. The fundamental command to achieve this is **route**.

Issuing the command:

**route**

Typically provides an output similar to the following:

#### ***Kernel IP routing table***

<b><i>Destination</i></b>	<b><i>Gateway</i></b>	<b><i>Genmask</i></b>	<b><i>Flags</i></b>	<b><i>Metric</i></b>	<b><i>Ref</i></b>	<b><i>Use</i></b>	<b><i>Iface</i></b>
<b><i>192.168.1.0</i></b>	<b><i>*</i></b>	<b><i>255.255.255.0</i></b>	<b><i>U</i></b>	<b><i>0</i></b>	<b><i>0</i></b>	<b><i>0</i></b>	<b><i>eth0</i></b>
<b><i>127.0.0.0</i></b>	<b><i>*</i></b>	<b><i>255.0.0.0</i></b>	<b><i>U</i></b>	<b><i>0</i></b>	<b><i>0</i></b>	<b><i>0</i></b>	<b><i>lo</i></b>
<b><i>default</i></b>	<b><i>gateway</i></b>	<b><i>0.0.0.0</i></b>		<b><i>UG</i></b>	<b><i>0</i></b>	<b><i>0</i></b>	<b><i>eth0</i></b>

From this output, we learn the following information:

<u>Destination</u>	This is network that we wish to go to. In our example we have:
192.168.1.0	This is our local network address, so anyone on this network is served by this line.
127.0.0.0	This is the local loopback address, that is one that we can only use to test our own IP stack. This loopback point is performed at the OSI model Network Layer.
Default	All other traffic will be sent to this address.
<u>Gateway</u>	This specifies the address of the device where information is to be forwarded to.
*	All information on the local network address is accepted.
Gateway	In this example, we see the name "gateway" because it has been defined in the /etc/hosts file. Otherwise we

	would see the actual IP address of the router to which we would be forwarding our packets to. (In this case, the IP address is 192.168.1.1 .)
<u>Genmask</u>	This is the subnet mask used to specify which traffic is to be routed on the specified destination.
255.255.255.0	All traffic on the local network is specified.
255.0.0.0	All loopback traffic is specified.
0.0.0.0	All remaining traffic is to use this network.
<u>Flags</u>	Specifies status / utilization of the interface.
	U (route is up)
	H (target is a host)
	G (use gateway)
	R (reinstate route for dynamic routing)
	D (dynamically installed by daemon or redirect)
	M (modified from routing daemon or redirect)
	A (installed by addrconf)
	C (cache entry)
	! (reject route)
<u>Metric</u>	Specifies which route should be used if more than one path is available to reach the remote location.
<u>References</u>	The number of references to this entry, but not utilized in Linux.
<u>Use</u>	A count of the lookups for this route.
<u>Interface</u>	The interface that is to be used for forwarding the information.

After we have displayed the routing table, we might find it necessary to manually modify the entries. Options include:

add	Add a new route to the table.
del	Delete an existing route from the table.
-target	Specify a network or host using an IP address.
-net	Specify a target network.
netmask	Specify the subnet mask needed for a network.
gw	Specify that the routing is to a gateway router. Static routes must be set up.

To understand how the routing table is used, we need to evaluate two situations – communicating with a another host on the same network and communicating with a host on a foreign network. To differentiate between the two, the TCP software compares the host address, destination address, and the subnet mask.

If the network address for the local host and destination host are the same, then the host local host obtains the MAC address of the destination host (using the ARP protocol) and transmits the data out of the local network card by looking up the interface in the routing table. In our example it is specified by the line:

```
192.168.1.0 * 255.255.255.0 U 0 0 0
eth0
```



We can observe that any IP address with the network address of 192.168.1.0 (subnet mask of 255.255.255.0) is to be transmitted on eth0.

If the network address for the destination host is different, then the data is forwarded to the network router, as specified by the line:

```
default      gateway      0.0.0.0      UG      0      0      0
eth0
```

This specifies that all other network addresses are to be (also) transmitted on eth0 in this example. If we had a WAN interface on our computer (making it a router), then it would specify a different interface.

### 11.3.1 Examples of usage

```
route add -net 127.0.0.0
```

Adds the normal loopback entry, using netmask 255.0.0.0 (class A net, determined from the destination address) and associated with the "lo" device (assuming this device was previously set up correctly with ifconfig(8)).

```
route add -net 192.56.76.0 netmask 255.255.255.0 dev eth0
```

Adds a route to the network 192.56.76.x via "eth0". The Class C netmask modifier is not really necessary here because 192.\* is a Class C IP address. The word "dev" can be omitted here.

```
route add default gw mango-gw
```

Adds a default route (which will be used if no other route matches). All packets using this route will be gatewayed through "mango-gw". The device which will actually be used for that route depends on how we can reach "mango-gw" - the static route to "mango-gw" will have to be set up before in the /etc/hosts file.

```
route add -net 224.0.0.0 netmask 240.0.0.0 dev eth0
```

This is an obscure one documented so people know how to do it. This sets all of the class D (multicast) IP routes to go via "eth0". This is the correct normal configuration line with a multicasting kernel.

## 11.4 Network Address Translation Router (NAT)

One of the functions of a router is to create a "private" network. This is a local network that typically utilizes one or more of the public domain IP Addresses for internal use, and has a dedicated private IP Address visible to the public.

How this works is for the router on the Internet side is to have a private IP address, and this address is translated into an internal address for the local network. Normally, anyone on the outside is not able to initiate a session with an internal host because they will be blocked, but a user on the inside can initiate a session to an external host. Thus a user can establish a connection to a remote web site to view a page.

### 11.4.1 IP Masquerading<sup>1,2</sup>

IP Masquerading is a method by which we can hide a local network from the rest or the Internet world. This method provides the first level of **firewall** protection.

The best way to describe the process is by example.

A client wishes to observe a web page for IP masquerading. The URL is typed into the browser – **www.ipmasq.cjb.net** .

Since we do not have the IP Address, we look to the client host table, but since it is not there, we request the address from the DNS server at our ISP, via our router.

The router receives the DNS request from the client. It then removes the client address, selects an unused service port (say 2001), and assigns the port number to the request – it then stores this mapping of client IP Address to port number in memory. Finally, the DNS request is transmitted to the ISP DNS Server with the origination address of the router. Hence the DNS server does not see the address of the origination client.

By whatever means, the DNS server obtains the IP address for ipmasq.cjb.net, and sends it back to the router address with port 2001.

The router then observes the port 2001, and realizes that this is the request that it transmitted previously, and retrieving the client address from the 2001 port map, the router strips its own address from the return address and forwards the packet to the originating client.

Now that the client has the IP Address for the URL, it then issues a request to the IP Address for the web page out to the router.

This process is repeated a number of times with the client's IP Address being deleted by the router, and the request sent out to the Internet with the router's IP Address.

IP Masquerading allows us to set up rules for routing our traffic. These are contained in the **/proc/net/ip\_fwchains** file. These rules provide the following options:

<b>ACCEPT</b>	Allows the packet to pass through the router.
<b>DENY</b>	Drops the packets without forwarding it.
<b>REJECT</b>	Drops the packet without forwarding, but sends a message back to the originator saying it was dropped.
<b>MASQ</b>	Packets are masqueraded as though they come from the client.

The above rules are set up on one of four different categories, as might be appropriate for the desired service:

IP Input Chain	Packets terminating to the local network
IP Output Chain	Packets originating from the local network
IP Forwarding Chain	Packets that are to be forwarded through the firewall
User Defined Chain	If the other rules don't work, use this category

<sup>1</sup> **Red Hat Linux Bible**; by Christopher Negus; IDG Books

<sup>2</sup> **Linux – The Complete Reference**; by Richard Petersen; Osborne / McGraw Hill

For additional information, research the manual pages for **ipchains** and **ipfw**, and the IP Masquerade mini-HOWTO. You can also go to the [www.ipmasq.cjb.net](http://www.ipmasq.cjb.net) web page.

#### 11.4.2 IP Forwarding (Not Complete)

After we set up Masquerading on a router, we commonly need to set up a process called IP Forwarding.

IP Forwarding is required to make a Linux gateway router operate.

To setup IP Forwarding, we need to modify a specific file. Using either vi or pico, modify the `/etc/sysconfig/network` file. The normal content of this file is:

```
NETWORKING = yes
FORWARD_IPV4 = no
HOSTNAME = {your full host name address (49.devry.edu)}
DOMAINNAME = {your domain name address (devry.edu)}
GATEWAY = {IP Address of the network Gateway router
          (205.205.205.111)}
GATEWAYDEV = eth0
```

We need to modify this to set the line **FORWARD\_IPV4 = yes**.

On the client host, we need to add the route to the gateway. Issue the command:

```
route add default gw {gateway IP Address}
```

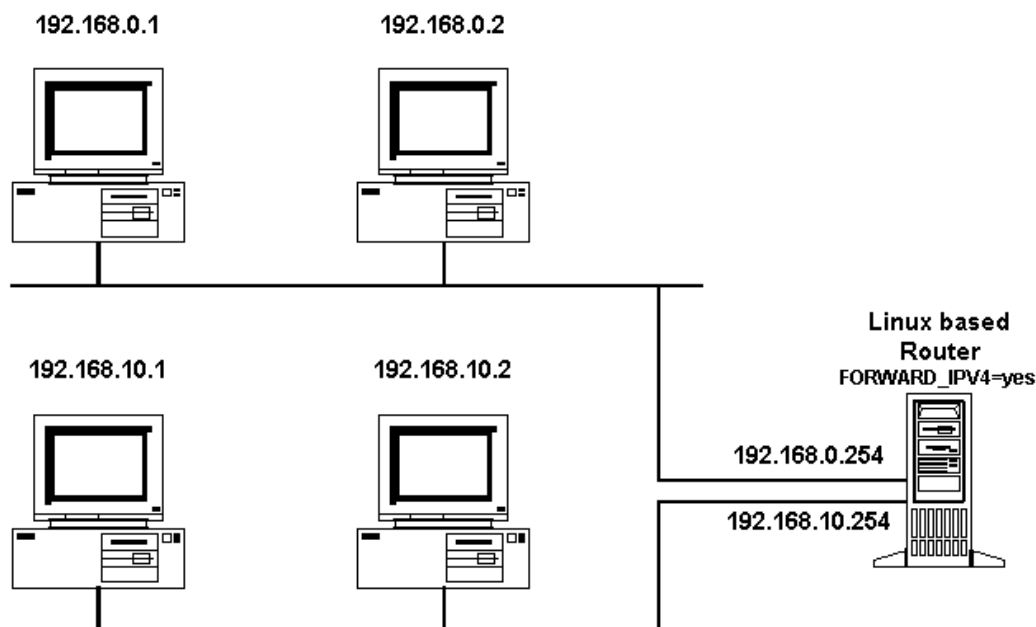


Figure 11.1: Simple Linux Router Network

## 11.5 IP Chains Firewall<sup>3,4,5</sup>

A Firewall Server is a router that protects all or part of a network from outside intrusion. There are two types of firewalls – **filtering** and **proxy**. Commonly, a firewall server is installed to protect a **private network**. In this concept, it is commonly referred to as a Network Address Translator, or NAT. This function is achieved by using IP Masquerading.

A filtering firewall looks at each entering packet and processes set of rules to decide if the packet will be allowed to pass through the server. As such it operates at the OSI Network Layer to decide if the packet is forwarded.

A NAT firewall performs a translation of the packet addresses by being an intermediary to the flow of packets. In this way, all traffic originated from within the private network appears as if it originated from the NAT server, which converts the originating host address to its own and forwards the packet to the outside. Therefore, the firewall operates at the OSI Application Layer.

Two types of private networks can be constructed, either a direct connected one or one that is behind a firewall. The latter establishes a **Demilitarized Zone**, or **DMZ**, which is the **Firewall Network**.

IP Filtering works by testing each packet against various rules that the administrator creates. Red Hat Linux comes with a package called **ipchains**, and the **diald** program that can be used to set up the rules. This type of firewall is relatively easy to configure, is efficient, and can be installed on a 386 / 486 Intel processor which can support a DS-1 / T-1 interconnection to the network. One problem with the IP Filter firewall is that it is subject to penetration by crackers who may attempt to gain access. Tools have been created to support testing of the filter firewall.

The filtering process works by setting up a set of **chains** and **rules**. Each packet must pass through each chain “Link”, where each link is a set of rules that each packet is tested against. A packet enters an interface, is tested to see if it is allowed, then it is tested to see if it is allowed to be forwarded, and finally passed to the output for evaluation against that set of rules.

Proxy firewalls require a password to gain access to the private network, making them relatively difficult to pass through when set up properly. Proxy servers are more difficult to set up, operate inefficiently and are vulnerable to software bugs and subject to new attack methods. Configuration of the proxy server requires each application to be configured in order to be utilized. Because of the address translation, doubling the addressing process, it is inefficient. Unfortunately when a hole is discovered in the firewall, it must be fixed via the programming of the proxy itself. Most administrators are not capable of the level of control.

### 11.5.1 Firewall Background

Before we can really explain how the firewall works, we need to review some of the terms and operations which make up each **rule**. The basic command structure of the rule is:

---

<sup>3</sup> **Red Hat Linux 6 in Small Business**, Paul Sery and Eric Harper, M&T / IDG Books

<sup>4</sup> **Building Linux and OpenBSD Firewalls**, Wes Sonnenreich, Tom Yates; John Wiley & Sons;

<sup>5</sup> Point Connect Networks; clarkconnect.org

## **ipchains *action chain condition policy***

Rules allow the administrator to provide additional guidelines by which each packet is evaluated. This can be as simple or complicated as the administrator desires. Keep in mind that the more defined the rules are, the more secure the firewall is. Our discussion here will be relatively simple and not present a fully secured system.

Each of the variables (action, chain, condition and policy) are covered in more detail below.

### **11.5.1.1 Action**

The actions which may be imposed in establishing our rules are as follows:

- F** Flush or erase all previous chain rules
- x** Flush out all existing chains
- P** Set a policy for a specific chain
- L** List the rules previously established
- v** Display more (verbose) information about the rules. It is used in conjunction with the **-L** option.
- n** Represent IP Addresses in the dotted numeric notation (not URL)
- A** Append another rule to a previously set policy
- D** Delete the specified rule from the chain
- R** Replace a specified rule of the chain with another
- I** Insert a rule at the specified location
- N** Create a new chain
- h** Help

### **11.5.1.2 Chain**

The filtering process works by setting up a series of **chains** and **rules**. Each packet must pass through each chain, of which there are three. These are:

1. **input**
2. **forward**
3. **output**

#### **11.5.1.2.1 Input**

Input chains accept data from an incoming interface. They determine if the packet is allowed to enter.

#### **11.5.1.2.2 Forward**

Forward chains provide for the modification of a packet, normally used for Masquerading.

#### **11.5.1.2.3 Output**

The output chain determines if data is allowed to be sent out of an interface.

Chains are not responsible for the routing of a packet – that is the responsibility of the routing tables. Chains are only responsible in determining whether packets may be routed.

A packet which is Denied is just dropped into the “bit bucket”, never to be seen or heard from again. This is the most efficient process, but leaves no feedback to the originator.

#### 11.5.1.2.4 Condition

Conditions establish that is to be tested. There may be multiple conditions for a given rule. The conditions are:

- p** Specifies a protocol type, which include:  
tcp  
udp  
icmp  
all
- b** A specified rule is bi-directional (only have to write it once). This is not applicable to a SNY rule.
- s** Specifies the Source IP Address
- i** Specifies the specific interface, typical interfaces include:  
eth0(1) ethernet port  
ppp0(1) dialup ppp modem port
- y** Specifies that allows you to match to a matched **SYN** packet <sup>6</sup>
- !** The previous specified condition is “negated” or reversed
- dport** Specifies the Destination Address of the packet (note double dash)
- sport** Specifies the Source Address of the packet (note double dash)
- j** Specifies the target (this condition is always last)

#### 11.5.1.2.5 Policy

Each chain consists of a set of rules which determine if the packet is to die or continue. The action is referred to as a policy or target. These are:

1. **ACCEPT**
2. **DENY**
3. **REJECT**
4. **MASQUerade**
5. **REDIRECT**
6. **RETURN**

#### 11.5.1.2.6 Accept

The rule condition allows the packet to continue through the chain and onto the next link or chain.

#### 11.5.1.2.7 Deny

This rule condition kills the packet and does not inform the originator. This is the most efficient process, requiring minimum cpu processing, but leaves the originator not knowing what happened (this really may be a good thing).

---

<sup>6</sup> SYN Flag – When host on the Internet attempts to respond to a TCP connection request that was initiated by a local network user. In order to establish a session, the SYN packet must be allowed, but we do not want to allow unjustified SYN packets which are not in response to our request.

#### 11.5.1.2.8 Reject

This rule condition kills the packet, and then informs the originator that they did not succeed. A Rejected ICMP packet is also not allowed to pass through the link, but an ICMP message is returned back to the originator, advising them of the failure. It is better to know that you were rejected, but it does require additional process power. Another issue that must be considered in using a Reject is respect to the **cracker**. A cracker can then continue to attempt probing to see if entrance can be obtained. Hence it is better to Deny service for security purposes.

#### 11.5.1.2.9 Masquerade

Masquerading is something like a proxy service, but is much easier to configure and operate. One of the greatest benefits is in the creation of a private network – which is very common in a home or small business network.

The Masquerade may only be used in the Forward chain. It may be used only if the Kernel has been configured with CONFIG\_IP\_MASQUERADE set to true in the Kernel (this is the default status in Red Hat). Packets matching the rule will have the source IP Address modified to appear as if it were on the other network. This target is extremely useful for hiding a private network from the rest of the Internet.

#### 11.5.1.2.10 Redirect

Redirect is used exclusively within an input chain. In order to work, the Kernel must be configured with the CONFIG\_IP\_PROXY set to true (this is the default status in Red Hat). Packets which match the condition for this rule are re-routed to a local socket, even if they were to go somewhere else. The benefits are limited except in specific instances, such as the Transparent Web Cache.

#### 11.5.1.2.11 Return

Return may be used in any chain, but is useful generally in user defined chains. When used, the default policy for the chain determines the fate of the packet that matches the specific rule.

### 11.5.1.3 Setting up a Filtering Firewall

In order to set up IP filtering, we need to install the Red Hat package **ipchains**. First test to see if it has been installed during the initial installation with:

```
rpm -q ipchains
```

If not, issue the command:

```
rpm -ivh /mnt/cdrom/RedHat/RPMS/ipchains*
```

#### 11.5.1.3.1 Simple Filter

After logging on the server as the root administrator, issue the command:

```
ipchains -L -n
```

You should observe:

```
Chain input (policy ACCEPT)
Chain forward (policy ACCEPT)
Chain output (policy ACCEPT)
```

Hence we see that initially there are no rules set, and everything is accepted.

Before we get started, issue the command:

```
ipchains -F
```

Now issue the following commands:

```
ipchains -P input DENY
ipchains -P forward DENY
ipchains -P output DENY
```

All packets that try to transverse the firewall will be denied – great if you don't want to connect to anything.

One of the first things we want to do from inside our private network is to browse the Internet with our web browser. Issue the command:

```
ipchains -A output -p tcp -i ppp0 -d 0.0.0.0/0 www -j ACCEPT
ipchains -A input -p tcp -i ppp0 ! -y -d 0.0.0.0/0 www -j
ACCEPT
```

1. The source and destination addresses are 0.0.0.0/0, which indicate that the whole Internet with zero 1's in the Subnet mask.
2. Only TCP packets are allowed to pass through the firewall.
3. The ! -y specifies that only packets from the outside that are in response to internal packets are allowed to pass, and have had their SYN bit set.
4. The specified interface here is ppp0. If you are setting up your interface for DSL or cable modem, you need to set the interface to either eth0 or eth1, whichever is connected to the modem.

Issue the command:

```
ipchains -L -n
```

you should observe:

```
Chain input (policy DENY)
target    prot  opt  source      destination ports
ACCEPT    tcp  !y--- 0.0.0.0/0    0.0.0.0/0    * -> 80
Chain forward (policy DENY)
Chain output (policy DENY)
target    prot  opt  source      destination ports
ACCEPT    tcp  ----- 0.0.0.0/0    0.0.0.0/0    * -> 80
```



Now we want to allow DNS queries to go out. These are connectionless sessions, so a UDP connection is required. Issue the commands:

```
ipchains -A output -p UDP -i ppp0 -d 0.0.0.0/0 domain -j
ACCEPT
ipchains -A input -p UDP -i ppp0 -d 0.0.0.0/0 domain -j
ACCEPT
```

1. The first rule sets the UDP protocol of the type domain.
2. The second rule allows the returning response packets to pass back with the domain information.

Again issue the command:

```
ipchains -L -n
```

and we observe:

```
Chain input (policy DENY)
Target      prot  opt  source          destination ports
ACCEPT tcp  !y    0.0.0.0/0      0.0.0.0/0      *-> 80
ACCEPT udp  ----- 0.0.0.0/0      0.0.0.0/0      *-> 53
Chain forward (policy DENY)
Chain output (policy DENY)
ACCEPT tcp  ----- 0.0.0.0/0      0.0.0.0/0      *-> 80
ACCEPT udp  ----- 0.0.0.0/0      0.0.0.0/0      *-> 53
```

Now lets work with the ICMP protocol.

```
ipchains -F input
ipchains -A input -p tcp -j REJECT
```

In this case when we telnet, the ICMP message is killed, but the originator will get the message “connection refused”.

### 11.5.2 A Working Filter

When you build a full filter, two approaches may be used. The first method is to open up your firewall and then deny undesirable services, or we can deny everything and then allow only desired services. The latter approach is the preferred (and safer) method, which we will illustrate here.

This script is taken from a firewall system called “ClarkConnect.org”. The firewall installation that they provide is available for free for the downloading off of the Internet from **clarkconnect.org**. It is designed to provide a very secure firewall for a local cable / DSL modem environment in your home. It references additional features that are not necessarily covered in this document. This is provided as an example and for your review (with permission of Clark Connect), and is not to be copied into another firewall that is not using PointClark Networks software. (This listing is not the latest version and may have updates made to it.) It is shown here for illustrative purposes only. (Note that some lines are longer than the width of the page.)

```
#!/bin/sh
```

```
#####
#
# Written by Dinesh Kandiah
# Copyright (C) 2000 Point Clark Networks
#
# This program is free software; you can redistribute it and/or
# modify it under the terms of the GNU General Public License
# as published by the Free Software Foundation; either version 2
# of the License, or (at your option) any later version.
#
# This program is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
# GNU General Public License for more details.
#
# You should have received a copy of the GNU General Public License
# along with this program; if not, write to the Free Software
# Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA 02111-1307, USA.
#
# Inspired by...
# - Trinity OS
# - Linux Security
# - linux-firewall-tools.com
# - Usenet
#
#####
#
# This configuration assumes the following setup:
#
# 1) The external interface is running on "ppp0" or "eth0"
# 2) The external IP address is dynamically assigned
# 3) The internal IP Masqueraded network interface is "eth1"
#
#####

#-----
# Setup
#-----

# Interfaces

LOOPBACKIF="lo"
LOOPBACKIP="127.0.0.1"
EXTIF="eth0"
INTIF="eth1"
INTLAN="192.168.1.0/24"

# Double check our Internet connection
if [ $EXTIF = "ppp0" ]; then
    CHECK=`/sbin/ifconfig | grep ppp0 | awk '{ print $1 }'`
    if [ -z $CHECK ]; then
        echo "PPP Internet connection is down... exiting"
        # We do not have an Internet connection!
        # Send a broadcast message
        exit
    fi
fi

if [ $EXTIF = "eth0" ]; then
    CHECK=`/sbin/ifconfig | grep eth0 | awk '{ print $1 }'`
    if [ -z $CHECK ]; then
        echo "Internet connection is down... exiting"
        # We do not have an Internet connection!
        # Send a broadcast message
        exit
    fi
fi
```

```

# IP Mask for all IP addresses
UNIV="0.0.0.0/0"

# IP Mask for broadcast transmissions
BROADCAST="255.255.255.255"

# IP address of the external interface
EXTIP=`/sbin/ifconfig | grep -A 4 $EXTIF | awk '/inet/ { print $2 }' | sed
-e s/addr://`

# Broadcast address of the external network
case "$EXTIF" in
ppp0)
EXTBROAD=`/sbin/ifconfig | grep -A 1 $EXTIF | awk '/inet/ { print $3 }' |
sed -e s/P-t-P://`
;;
eth0)
EXTBROAD=`/sbin/ifconfig | grep -A 1 $EXTIF | awk '/Bcast/ { print $3 }' |
sed -e s/Bcast://`
;;
esac

# Specification of the high unprivileged IP ports.
UNPRIVPORTS="1024:65535"

# Specification of X Windows System (TCP) ports:
XWINDOWS_PORTS="6000:6010"

# Logging state
#LOG="-l"
LOG=" "

#-----
# Default Policies
#-----
ipchains -P input REJECT
ipchains -P output REJECT
ipchains -P forward REJECT

# Flushing all old rules and setting all default policies to REJECT
ipchains -F input
ipchains -F output
ipchains -F forward

#-----
# Masquerading Timeouts
#-----
# Set timeout values for masq sessions (seconds).
#
# Item #1 - 2 hrs timeout for TCP session timeouts
# Item #2 - 10 sec timeout for traffic after the TCP/IP "FIN" packet is
received
# Item #3 - 60 sec timeout for UDP traffic
#
ipchains -M -S 7200 10 60

#-----
# General
#-----

# Enable IP Forwarding, if it isn't already
sysctl -w net.ipv4.ip_forward=1 >/dev/null

# Disable IP spoofing attacks.
sysctl -w net.ipv4.conf.all.rp_filter=1 >/dev/null

```

```

# Enable always defragging
sysctl -w net.ipv4.ip_always_defrag=1 >/dev/null

# Enable TCP SYN Cookie protection:
sysctl -w net.ipv4.tcp_syncookies=1 >/dev/null

# Enabling dynamic TCP/IP address hacking.
sysctl -w net.ipv4.ip_dynaddr=1 >/dev/null

# Disable ICMP broadcast echo protection
sysctl -w net.ipv4.icmp_echo_ignore_broadcasts=1 >/dev/null

# Enable bad error message protection
sysctl -w net.ipv4.icmp_ignore_bogus_error_responses=1 >/dev/null

# Disable ICMP Re-directs
sysctl -w net.ipv4.conf.all.accept_redirects=0 >/dev/null
sysctl -w net.ipv4.conf.all.send_redirects=0 >/dev/null

# Ensure that source-routed packets are dropped
sysctl -w net.ipv4.conf.all.accept_source_route=0 >/dev/null

# Log spoofed, source-routed, and redirect packets
sysctl -w net.ipv4.conf.all.log_martians=1 >/dev/null

#-----
# Masq Modules
#-----
/sbin/modprobe ip_masq_autofw
/sbin/modprobe ip_masq_cuseeme
/sbin/modprobe ip_masq_ftp
/sbin/modprobe ip_masq_irc
/sbin/modprobe ip_masq_portfw
/sbin/modprobe ip_masq_quake
/sbin/modprobe ip_masq_raudio
/sbin/modprobe ip_masq_vdolive
/sbin/modprobe ip_masq_user
/sbin/modprobe ip_masq_icq

#####
#                                     #
#                                     #   Input Rules   #
#                                     #
#####

#-----
# Incoming Traffic on Internal LAN
#-----

# DHCP server
ipchains -A input -j ACCEPT -i $INTIF -p udp -s $UNIV bootpc -d $BROADCAST/0
bootps
ipchains -A input -j ACCEPT -i $INTIF -p tcp -s $UNIV bootps -d $BROADCAST/0
bootps

#-----
# Incoming Traffic from the External Interface
#-----
# Explicitly allow certain services

# Dynamic IP address for ADSL or cable modem connection enabled.
ipchains -A input -j ACCEPT -i $EXTIF -p udp -s $UNIV bootps -d $BROADCAST/0
bootpc
ipchains -A input -j ACCEPT -i $EXTIF -p tcp -s $UNIV bootps -d $BROADCAST/0
bootpc

# ICMP: Allow ICMP packets from all external TCP/IP addresses.

```

```

ipchains -A input -j ACCEPT -i $EXTIF -p icmp -s $UNIV -d $EXTIP

# HTTP: Allow external users to connect to HTTP services.
ipchains -A input -j ACCEPT -i $EXTIF -p tcp -s $UNIV -d $EXTIP www

# SSH server: Allow external computers to connect to SSH services.
ipchains -A input -j ACCEPT -i $EXTIF -p tcp -s $UNIV -d $EXTIP ssh

# PortSentry: We let PortSentry find people scanning the system
ipchains -A input -j ACCEPT -i $EXTIF -p tcp -s $UNIV -d $EXTIP 1
ipchains -A input -j ACCEPT -i $EXTIF -p udp -s $UNIV -d $EXTIP 1
ipchains -A input -j ACCEPT -i $EXTIF -p tcp -s $UNIV -d $EXTIP 11
ipchains -A input -j ACCEPT -i $EXTIF -p tcp -s $UNIV -d $EXTIP 15
ipchains -A input -j ACCEPT -i $EXTIF -p tcp -s $UNIV -d $EXTIP 79
ipchains -A input -j ACCEPT -i $EXTIF -p tcp -s $UNIV -d $EXTIP 111
ipchains -A input -j ACCEPT -i $EXTIF -p tcp -s $UNIV -d $EXTIP 119

# Point Clark Networks: if you are going to use your firewall as
# a mail server or FTP server, then uncomment the appropriate lines
# below. You will need to uncomment the output rules further below!
# You will also need to edit the /etc/hosts.allow and /etc/inetd.conf.

# SMTP & FTP:
#ipchains -A input -j ACCEPT -i $EXTIF -p tcp -s $UNIV -d $EXTIP smtp
#ipchains -A input -j ACCEPT -i $EXTIF -p tcp -s $UNIV -d $EXTIP ftp
#ipchains -A input -j ACCEPT -i $EXTIF -p tcp -s $UNIV -d $EXTIP ftp-data

#-----
# Specific Input Rejections on the EXTERNAL interface
#-----

# IP spoofing and private/reserved networks
ipchains -A input -j REJECT -i $EXTIF -s $INTLAN -d $UNIV $LOG
ipchains -A input -j REJECT -i $EXTIF -s $LOOPBACKIP $LOG

# Private and reserved networks
ipchains -A input -j REJECT -i $EXTIF -s 0.0.0.0/8 -d $UNIV $LOG
ipchains -A input -j REJECT -i $EXTIF -s 1.0.0.0/8 -d $UNIV $LOG
ipchains -A input -j REJECT -i $EXTIF -s 10.0.0.0/8 -d $UNIV $LOG
ipchains -A input -j REJECT -i $EXTIF -s 23.0.0.0/8 -d $UNIV $LOG
ipchains -A input -j REJECT -i $EXTIF -s 31.0.0.0/8 -d $UNIV $LOG
ipchains -A input -j REJECT -i $EXTIF -s 96.0.0.0/3 -d $UNIV $LOG
ipchains -A input -j REJECT -i $EXTIF -s 128.0.0.0/16 -d $UNIV $LOG
ipchains -A input -j REJECT -i $EXTIF -s 128.9.64.26/32 -d $UNIV $LOG
ipchains -A input -j REJECT -i $EXTIF -s 128.66.0.0/16 -d $UNIV $LOG
ipchains -A input -j REJECT -i $EXTIF -s 172.16.0.0/12 -d $UNIV $LOG
ipchains -A input -j REJECT -i $EXTIF -s 191.255.0.0/16 -d $UNIV $LOG
ipchains -A input -j REJECT -i $EXTIF -s 192.0.0.0/16 -d $UNIV $LOG
ipchains -A input -j REJECT -i $EXTIF -s 192.168.0.0/16 -d $UNIV $LOG
ipchains -A input -j REJECT -i $EXTIF -s 197.0.0.0/16 -d $UNIV $LOG
ipchains -A input -j REJECT -i $EXTIF -s 201.0.0.0/8 -d $UNIV $LOG
ipchains -A input -j REJECT -i $EXTIF -s 223.255.255.0/24 -d $UNIV $LOG
ipchains -A input -j REJECT -i $EXTIF -s 240.0.0.0/5 -d $UNIV $LOG
ipchains -A input -j REJECT -i $EXTIF -s 248.0.0.0/5 -d $UNIV $LOG

# SMB and CIFS: Reject SMB and CIFS traffic FROM and TO external machines.
ipchains -A input -j REJECT -i $EXTIF -p tcp -s $UNIV -d $EXTIP 137
ipchains -A input -j REJECT -i $EXTIF -p udp -s $UNIV -d $EXTBROAD 137
ipchains -A input -j REJECT -i $EXTIF -p tcp -s $UNIV -d $EXTBROAD 137
ipchains -A input -j REJECT -i $EXTIF -p tcp -s $UNIV -d $EXTIP 138
ipchains -A input -j REJECT -i $EXTIF -p udp -s $UNIV -d $EXTBROAD 138
ipchains -A input -j REJECT -i $EXTIF -p tcp -s $UNIV -d $EXTBROAD 138
ipchains -A input -j REJECT -i $EXTIF -p tcp -s $UNIV -d $EXTIP 139
ipchains -A input -j REJECT -i $EXTIF -p udp -s $UNIV -d $EXTBROAD 139
ipchains -A input -j REJECT -i $EXTIF -p tcp -s $UNIV -d $EXTBROAD 139
ipchains -A input -j REJECT -i $EXTIF -p tcp -s $UNIV -d $EXTIP 445

```

```

ipchains -A input -j REJECT -i $EXTIF -p udp -s $UNIV -d $EXTBROAD 445
ipchains -A input -j REJECT -i $EXTIF -p tcp -s $UNIV -d $EXTBROAD 445
ipchains -A input -j REJECT -i $EXTIF -p tcp -s $UNIV 137 -d $EXTIF
ipchains -A input -j REJECT -i $EXTIF -p tcp -s $UNIV 138 -d $EXTIF
ipchains -A input -j REJECT -i $EXTIF -p tcp -s $UNIV 139 -d $EXTIF
ipchains -A input -j REJECT -i $EXTIF -p tcp -s $UNIV 445 -d $EXTIF

# NFS: Reject NFS traffic from and to external machines.
ipchains -A input -j REJECT -i $EXTIF -p tcp -s $UNIV -d $EXTIF 2049
ipchains -A input -j REJECT -i $EXTIF -p tcp -s $UNIV 2049 -d $EXTIF

#-----
# Incoming Traffic on all Interfaces
#-----

# AUTH: Allow the authentication protocol, but disable it in
/etc/inetd.conf.
# (For legacy TCP/IP stack issues).
ipchains -A input -j ACCEPT -p tcp -s $UNIV -d $UNIV auth
ipchains -A input -j ACCEPT -p tcp -s $UNIV auth -d $UNIV

#-----
# Incoming Traffic on the Internal LAN
#-----

# Local interface, local machines, going anywhere is valid.
ipchains -A input -j ACCEPT -i $INTIF -s $INTLAN -d $UNIV

# Loopback interface is valid.
ipchains -A input -j ACCEPT -i $LOOPBACKIF -s $UNIV -d $UNIV

# HIGH PORTS:
# Enable all high unprivileged ports for all reply TCP/UDP traffic
ipchains -A input -j ACCEPT ! -y -p tcp -s $UNIV -d $EXTIF $UNPRIVPORTS
ipchains -A input -j ACCEPT -p tcp -s $UNIV ftp-data -d $EXTIF $UNPRIVPORTS
ipchains -A input -j ACCEPT -p udp -s $UNIV -d $EXTIF $UNPRIVPORTS

#-----
-
# Catch All Input Rule
#-----
ipchains -A input -j REJECT -s $UNIV -d $UNIV $LOG

#####
#                                     #
#                                     #
#                                     #
#                                     #
#####

#-----
# Outgoing traffic on the Internal LAN
#-----
# Let everything out

# Local interface, any source going to local net is valid.
ipchains -A output -j ACCEPT -i $INTIF -s $UNIV -d $INTLAN

# Loopback interface is valid
ipchains -A output -j ACCEPT -i $LOOPBACKIF -s $UNIV -d $UNIV

#-----
# Outgoing Traffic on External Interface
#-----
# This will control what traffic can go out on the external interfaces.

# Reject outgoing traffic to the local net from the remote interface,

```

```

# stuffed routing; deny & log
ipchains -A output -j REJECT -i $EXTIF -s $UNIV -d $INTLAN $LOG

# Reject outgoing traffic from the local net from the external interface,
# stuffed masquerading, deny and log
ipchains -A output -j REJECT -i $EXTIF -s $INTLAN -d $UNIV $LOG

# DHCP:
ipchains -A output -j ACCEPT -i $EXTIF -p tcp -s $UNIV bootpc -d $UNIV
bootps
ipchains -A output -j ACCEPT -i $EXTIF -p udp -s $UNIV bootpc -d $UNIV
bootps

# Allowing output for SSH and Web
ipchains -A output -j ACCEPT -i $EXTIF -p tcp -s $EXTIF ssh -d $UNIV
$UNPRIVPORTS
ipchains -A output -j ACCEPT -i $EXTIF -p tcp -s $EXTIF www -d $UNIV
$UNPRIVPORTS

# Point Clark Networks: if you are going to use your firewall as
# a mail server or FTP server, then uncomment the appropriate lines
# below. You will need to do the same for the input rules above!
# You will also need to edit the /etc/hosts.allow and /etc/inetd.conf.

# SMTP & FTP:
#ipchains -A output -j ACCEPT -i $EXTIF -p tcp -s $UNIV smtp -d $UNIV
#ipchains -A output -j ACCEPT -i $EXTIF -p tcp -s $UNIV ftp -d $UNIV
#ipchains -A output -j ACCEPT -i $EXTIF -p tcp -s $UNIV ftp-data -d $UNIV

#-----
# Outgoing Traffic on all Interfaces
#-----
# Controls output traffic for all interfaces

# AUTH: Allow the authentication protocol, but disable it in
/etc/inetd.conf.
# (For legacy TCP/IP stack issues).
ipchains -A output -j ACCEPT -p tcp -s $UNIV auth -d $UNIV
ipchains -A output -j ACCEPT -p tcp -s $UNIV -d $UNIV auth

# ICMP: Allow ICMP traffic out
ipchains -A output -j ACCEPT -p icmp -s $UNIV -d $UNIV

#-----
# Specific Output Rejections
#-----
# Reject traffic that you do not want out of the system.

# Reject outgoing traffic to the local net from the remote interface,
# stuffed routing; deny & log
ipchains -A output -j REJECT -i $EXTIF -s $UNIV -d $INTLAN $LOG

# Reject outgoing traffic from the local net from the external interface,
# stuffed masquerading, deny and log
ipchains -A output -j REJECT -i $EXTIF -s $INTLAN -d $UNIV $LOG

# Samba
ipchains -A output -j REJECT -i $EXTIF -p udp -s $EXTIF -d $UNIV 137 $LOG
ipchains -A output -j REJECT -i $EXTIF -p udp -s $EXTIF -d $UNIV 138 $LOG
ipchains -A output -j REJECT -i $EXTIF -p udp -s $EXTIF -d $UNIV 139 $LOG
ipchains -A output -j REJECT -i $EXTIF -p udp -s $EXTIF 137 -d $UNIV 137
$LOG
ipchains -A output -j REJECT -i $EXTIF -p udp -s $EXTIF 138 -d $UNIV 138
$LOG
ipchains -A output -j REJECT -i $EXTIF -p udp -s $EXTIF 139 -d $UNIV 139
$LOG

```

```

# Mountd.
ipchains -A output -j REJECT -i $EXTIF -p udp -s $EXTIP -d $UNIV 635 $LOG
ipchains -A output -j REJECT -i $EXTIF -p udp -s $EXTIP 635 -d $UNIV $LOG

# Remote Winsock.
ipchains -A output -j REJECT -i $EXTIF -p tcp -s $EXTIP -d $UNIV 1745 $LOG
ipchains -A output -j REJECT -i $EXTIF -p udp -s $EXTIP -d $UNIV 1745 $LOG

# NFS
ipchains -A output -j REJECT -i $EXTIF -p tcp -s $EXTIP -d $UNIV 2049 $LOG
ipchains -A output -j REJECT -i $EXTIF -p tcp -s $EXTIP 2049 -d $UNIV $LOG
ipchains -A output -j REJECT -i $EXTIF -p udp -s $EXTIP -d $UNIV 2049 $LOG
ipchains -A output -j REJECT -i $EXTIF -p udp -s $EXTIP 2049 -d $UNIV $LOG

# PcAnywhere.
ipchains -A output -j REJECT -i $EXTIF -p tcp -s $EXTIP -d $UNIV 5631 $LOG
ipchains -A output -j REJECT -i $EXTIF -p udp -s $EXTIP -d $UNIV 5631 $LOG
ipchains -A output -j REJECT -i $EXTIF -p tcp -s $EXTIP -d $UNIV 5632 $LOG
ipchains -A output -j REJECT -i $EXTIF -p udp -s $EXTIP -d $UNIV 5632 $LOG

# Xwindows.
ipchains -A output -j REJECT -i $EXTIF -p tcp -s $EXTIP -d $UNIV
$XWINDOWS_PORTS $LOG
ipchains -A output -j REJECT -i $EXTIF -p udp -s $EXTIP -d $UNIV
$XWINDOWS_PORTS $LOG

# NetBsus.
ipchains -A output -j REJECT -i $EXTIF -p tcp -s $EXTIP -d $UNIV 12345 $LOG
ipchains -A output -j REJECT -i $EXTIF -p tcp -s $EXTIP -d $UNIV 12345 $LOG

# NetBus Pro.
ipchains -A output -j REJECT -i $EXTIF -p tcp -s $EXTIP -d $UNIV/0 20034
$LOG

# Back Orifice
ipchains -A output -j REJECT -i $EXTIF -p udp -s $EXTIP -d $UNIV/0 31337
$LOG

# Win Crash Trojan.
ipchains -A output -j REJECT -i $EXTIF -p tcp -s $EXTIP -d $UNIV/0 5742 $LOG

# Socket De Troye.
ipchains -A output -j REJECT -i $EXTIF -p tcp -s $EXTIP -d $UNIV/0 30303
$LOG

# Unkown Trojan Horse (Master's Paradise [CHR])
ipchains -A output -j REJECT -i $EXTIF -p tcp -s $EXTIP -d $UNIV/0 40421
$LOG

# Trinoo UDP flooder - Please note this port will probably change over time
ipchains -A output -j REJECT -i $EXTIF -p tcp -s $EXTIP 27665 -d $UNIV/0
$LOG
ipchains -A output -j REJECT -i $EXTIF -p udp -s $EXTIP 27444 -d $UNIV/0
$LOG
ipchains -A output -j REJECT -i $EXTIF -p udp -s $EXTIP 31335 -d $UNIV/0
$LOG

# Shat distributed flooder - Please note this will probably change over time
ipchains -A output -j REJECT -i $EXTIF -p tcp -s $EXTIP 20432 -d $UNIV/0
$LOG
ipchains -A output -j REJECT -i $EXTIF -p udp -s $EXTIP 18753 -d $UNIV/0
$LOG
ipchains -A output -j REJECT -i $EXTIF -p udp -s $EXTIP 20433 -d $UNIV/0
$LOG

#-----

```



```

# Allow all High Ports for return traffic.
#-----

ipchains -A output -j ACCEPT -p tcp -s $EXTIP $UNPRIVPORTS -d $UNIV
ipchains -A output -j ACCEPT -p udp -s $EXTIP $UNPRIVPORTS -d $UNIV

#-----
# Catch All Rule
#-----
# All other outgoing is denied and logged.
ipchains -A output -j REJECT -s $UNIV -d $UNIV $LOG

#####
#                                     #
#                               Forwarding                               #
#                                     #
#####

#-----
# Port Forwarding
#-----

# Uncomment this line if you want to port forward (flushes the portw table).
#ipmasqadm portfw -f

# Web
#
# Forward web traffic to an internal server (where xxx is the target IP).
# You will also need to shut down the webserver running on this machine
# as well as remove it from automatic startup, i.e.
# /etc/rc.d/init.d/httpd stop
# /sbin/chkconfig --del httpd
#
#ipmasqadm portfw -a -P tcp -L $EXTIP www -R 192.168.1.xxx www

# FTP
#
# Forward FTP traffic to an internal server (where xxx is the target IP).
# You will also need to change the /etc/hosts.allow and /etc/inetd.conf
# file.
#
#ipmasqadm portfw -a -P tcp -L $EXTIP ftp -R 192.168.1.xxx ftp
#ipmasqadm portfw -a -P tcp -L $EXTIP ftp-data -R 192.168.1.xxx ftp-data

# Other
#
# You can use the ipmasqadm tool for forwarding all sorts of other things
# (read: games!). Go to http://www.tsmservices.com/masq/ for details.

#-----
# Enable TCP/IP forwarding and masquerading from the Internal LAN
#-----

# Masquerade from local net on local interface to anywhere.
ipchains -A forward -j MASQ -i $EXTIF -s $INTLAN -d $UNIV

# Catch all rule, all other forwarding is denied.
ipchains -A forward -j REJECT -s $UNIV -d $UNIV $LOG

exit 0

```

#### end of script ####

## **11.6 IP Tables Firewall**

Since the original development of IP Chains, the process of setting up rules for limiting the forwarding of packets has been updated, and is now called **IP Tables**. IP Tables are quite similar to IP Chains in appearance, but does provide enhanced system security by adding increased filtration, flexibility, and statefull packet inspection. The benefit of IP Tables is that the same functionality may be accomplished as in IP Chains with less code. This is because the Kernel was improved to provide the enhanced security functions. What previously took 400 lines of code (shown previously), may now be accomplished in about 50 lines. Considerably shorter and more powerful.

IP Tables (iptables) is composed of two components, netfilter and iptables. The netfilter function operates in what is known as **kernel space**. The iptables function operates in what is known as **userspace**, and is used to set up, maintain, and display the rulebase that is used by netfilter.

The best way to illustrate the operation of IP Tables is through an example and explanation. We need to start with some definitions.

Chains are a set of three tables: Filter, Network Address Translation (NAT), and Mangle. Each table has a set of rules.

### **Filter**

A Filter is used to primarily to DROP or ACCEPT a packets based on their content, but does not alter the packets. Builtin chains are INPUT, FORWARD, and OUTPUT.

### **NAT**

The Network Address Table (NAT) processes packets that are modified by changing the address, such as used in masquerading, specifically for altering the source and destination address. Builtin chains include PREROUTING, OUTPUT, and POSTROUTING. They are used with DNAT, SNAT, and MASQUERADE targets only.

### **Destination NAT (DNAT)**

Used to alter the destination address of an IP address of an inbound packet from the Internet to a local host. DNAT is used to initially set up the address modification.

### **Source NAT (SNAT)**

Use to alter the source address of an IP address of an outbound packet from a host to the Internet. SNAT is used to initially set up the address modification.

### **Masquerade**

Differs from SNAT in that it only checks for an IP address to apply to outbound packets, most suitable for dynamic IP addresses.

**MANGLE**

This function is used to modify the Type of Service (TOS), Time To Live (TTL), and to mark fields in a packet. Builtin chains include PREROUTING and OUTPUT.

When a packet enters the router from either the local network or the Internet (or other network), it is first given several sanity checks, including a checksum verification. The packet is then forwarded to the PREROUTING (DNAT) chain, where the destination address is changed if necessary.

The next step is for the packet to be routed, depending upon its destination address. If the packet is destined for the local network, it is forwarded to the INPUT chain, where it may be filtered (accepted, dropped, or tested for against additional chains) or altered. Packets destined for the local network are forwarded to the OUTPUT POSTROUTING chains, while those that are not, are sent to the FORWARD and POSTROUTING chains, either of which allows each packet to again be filtered or altered.

**11.6.1 Using IPTables**

The IPTables may be set up from the command line through a series of commands. To a limited extent, they may also be set up using the X Windows Security Level Configuration Window, as it does not provide all of the flexibility available at the command line. The basic format of the command is:

```
# iptables [-t table-name] Command Chain Packet-Criteria Rule-
Criteria Target
```

**Table-name**

Specifies the name of the table that is to be operated on – FILTER, NAT, or MANGLE. If the Table-name is not specified, the default of FILTER is used.

**Command**

Specifies what is to do with the rest of the command line. These include:

- A Add one or more rules.
- I Inserts as a rule to the end of the chain.
- D Del rule to the specified location. If the location is not specified, then it is inserted at the beginning.
- R Replace the specified numbered rule with the designated new rule.
- L Display the existing rules in the chain.
  - v Verbose output, provides additional detail.
  - n Displays IP Address and Port number rather than names.
  - x Displays exact packet and byte counts instead of rounded values.
  - line-numbers Displays line numbers with each rule, used when modifying the rulebase.

- F Delete ALL rules from the chain. If specified chain is omitted, all rules of all chains are deleted.
- Z Change the value of all packet and byte counters to zero.
- X Delete a user defined chain.
- P Set the default target or policy for a builtin chain. The policy is applied to packets that do not match an existing rule in the specified chain. For a chain without a policy, unmatched packets are ACCEPTed.
- E Renames a chain.
- h Displays a help screen for the syntax of the iptables command.

### Chain

Designates the specific chain that is to be processed, FILTER, NAT, or MANGLE.

### Patch-Criteria

Specifies the match criteria that is to be applied to packets.

- p Match specified protocol.
- s Match specified source address.
- d Match specified destination address.
- i Match the specified input interface for the criteria. Used for INPUT, FORWARD, and PREROUTING chains.
- o Match the specified output interface for the criteria. Used for FORWARD, OUTPUT, and POSTROUTING chains.
- f Matches fragmented packets to specified criteria, as they do not contain the source or destination address, or other specified rules.
- j Specifies the jump target of the rule. That is, what is to happen to the packet if the criteria is matched.

### Rule-Criteria

Specifies rules that are to be applied.

- p Loads specified protocol module to match packets with specified protocol.
- tcp Specifies the TCP protocol.
  - dport Matches the destination port number or service name (/etc/services file)
  - sport Matches the source port number or service name.
  - syn Matches packets with the SYN bit set and bot the ACK and FIN bits cleared. It is equivalent to --tcp-flags SYN,RST,ACK,SYN
  - tcp-flag Specifies the TCP flag settings that constitute a match. Valid flags are SYN, ACK, FIN, RST, URG, PSH, ALL, NONE.

	--tcp-option	Matches TCP options based on the decimal value.
udp		Specifies the UDP protocol. Options are identical to TCP.
icmp		Specifies icmp type packets (ping and network status messages).
	--icmp-name	Matches icmp packets of the type named. To determine the types, issue the command: <b>iptables -p icmp -h</b>
	--state	Matches the packet against the specified state.
ESTABLISHED		A connection that has been established.
INVALID		A stateless or unidentifiable packet.
NEW		A new connection packet, normally specified by a SYN packet.
RELATED		Any packet exchange that has been ESTABLISHED.

### Target

		Specifies what happens to the packet, depending upon the matching against one of the rules.
ACCEPT		Continues to process the packet.
DNAT		Rewrites the destination address of a packet.
	--to-destination	Specifies the IP Address and Port (if applicable).
SNAT		Rewrites the source address of a packet.
	--to-source	Specifies the IP Address and Port (if applicable).
DROP		Kills the packet, without notice to the originator.
LOG		Logs specified type of packet types.
	--og-level	Specifies logging level as specified in /etc/syslog.conf.
	--log-prefix	Prefixes log entries with the specified string.
	--log-tcp-options	Logs options from the TCP packet header.
	--log-ip-options	Logs options from the IP packet header.
MASQUERADE		Similar to SNAT with the --to-source.
REJECT		Kills the packet, with notification to the originator.
RETURN		Terminates this chain and returns packet to originating chain.

Additional options are available, one should refer to the man page.

#### 11.6.2 A Working IP Table Filter

The following listing, provided by Clark Connect of an operation system, is a working configuration of an IP Table.

```
# iptables -L --line-numbers
Chain INPUT (policy DROP)
num target      prot opt source                destination
1  DROP          all  -- c-67-186-31-47.hsdl.pa.comcast.net  anywhere
2  ACCEPT        icmp -- anywhere              anywhere  icmp echo-reply
3  ACCEPT        icmp -- anywhere              anywhere  icmp destination-unreachable
4  ACCEPT        icmp -- anywhere              anywhere  icmp time-exceeded
5  ACCEPT        icmp -- anywhere              anywhere  icmp echo-request
6  DROP          icmp -- anywhere              anywhere
7  ACCEPT        udp  -- anywhere              anywhere  udp dpt:domain
8  ACCEPT        udp  -- anywhere              anywhere  state ESTABLISHED udp spt:domain
9  DROP          udp  -- anywhere              anywhere  udp spt:domain
10 DROP          all  -- anywhere              anywhere  state INVALID
11 REJECT        tcp  -- anywhere              anywhere  tcp flags:SYN,ACK/SYN,ACK state NEW reject-with tcp-reset
12 DROP          tcp  -- anywhere              anywhere  tcp flags:!SYN,RST,ACK/SYN state NEW
13 ACCEPT        all  -- anywhere              anywhere
14 ACCEPT        all  -- anywhere              anywhere
15 drop-reserved all  -- 127.0.0.0/8            anywhere
16 drop-reserved all  -- 2.0.0.0/8              anywhere
17 drop-reserved all  -- 96.0.0.0/3             anywhere
18 drop-reserved all  -- 169.254.0.0/16         anywhere
19 drop-reserved all  -- 223.0.0.0/8            anywhere
20 drop-reserved all  -- BASE-ADDRESS.MCAST.NET/4 anywhere
21 drop-reserved all  -- 240.0.0.0/4            anywhere
22 ACCEPT        udp  -- anywhere 67.187.114.236  udp spt:bootpc dpt:bootpc
23 ACCEPT        tcp  -- anywhere 67.187.114.236  tcp spt:bootpc dpt:bootpc
24 ACCEPT        tcp  -- anywhere 67.187.114.236  tcp dpt:ftp-data
25 ACCEPT        tcp  -- anywhere 67.187.114.236  tcp dpt:ftp
26 ACCEPT        tcp  -- anywhere 67.187.114.236  tcp dpt:ssh
27 ACCEPT        tcp  -- anywhere 67.187.114.236  tcp dpt:http
28 ACCEPT        tcp  -- anywhere 67.187.114.236  tcp dpt:1875
29 ACCEPT        udp  -- anywhere 67.187.114.236  udp dpts:1024:65535 state RELATED,ESTABLISHED
30 ACCEPT        tcp  -- anywhere 67.187.114.236  tcp dpts:1024:65535 state RELATED,ESTABLISHED
31 DROP          all  -- anywhere              anywhere

Chain FORWARD (policy DROP)
num target      prot opt source                destination
1  DROP          all  -- c-67-186-31-47.hsdl.pa.comcast.net  anywhere
2  ACCEPT        all  -- anywhere              anywhere
3  ACCEPT        all  -- anywhere              anywhere  state RELATED,ESTABLISHED
4  DROP          all  -- anywhere              anywhere

Chain OUTPUT (policy DROP)
num target      prot opt source                destination
1  ACCEPT        icmp -- anywhere              anywhere
2  ACCEPT        all  -- anywhere              anywhere
3  ACCEPT        all  -- anywhere              anywhere
4  ACCEPT        tcp  -- 67.187.114.236        anywhere  tcp spt:bootpc dpt:bootpc
5  ACCEPT        udp  -- 67.187.114.236        anywhere  udp spt:bootpc dpt:bootpc
6  ACCEPT        tcp  -- 67.187.114.236        anywhere  tcp spt:ftp-data
7  ACCEPT        tcp  -- 67.187.114.236        anywhere  tcp spt:ftp
8  ACCEPT        tcp  -- 67.187.114.236        anywhere  tcp spt:ssh
9  ACCEPT        tcp  -- 67.187.114.236        anywhere  tcp spt:http
10 ACCEPT        tcp  -- 67.187.114.236        anywhere  tcp spt:1875
11 ACCEPT        all  -- 67.187.114.236        anywhere
12 DROP          all  -- anywhere              anywhere

Chain drop-lan (0 references)
num target      prot opt source                destination
1  DROP          all  -- anywhere              anywhere

Chain drop-reserved (7 references)
num target      prot opt source                destination
1  DROP          all  -- anywhere              anywhere
```

## 11.X Commands Used in this Chapter

## 11.Y Chapter Review Questions



## Chapter Index

	C			
chains	12	IPChains		
	D	A Working Filter		17
Demilitarized Zone	12	Accept		10
dmesg	3	Action		13
Dual Ethernet	3	Chain		13
Dual Ethernet Router	4	Accept		14
Dynamic Routing	5	Condition		14
	E	Deny		14
Ethernet HOWTO	3	Forward		13
	F	Input		13
File		Masquerade		15
/etc/conf.modules	3	Output		13
/etc/lilo.conf	4	Policy		14
/etc/services	28	Redirect		15
/etc/sysconfig/network	11	Reject		15
/etc/syslog.conf	29	Return		15
/proc/net/ip_fwchains	10	Deny		10
Firewall		Masq		10
filtering	12	Reject		10
Firewall Background	12	Setting up a Filtering Firewall		15
Firewall Network	12	iptables		26
FORWARD_IPV4	11		K	
	G	Kernelspace		26
Gateway Address	4		M	
	I	Masquerading		10
Installation of Two NIC Cards	5	modprobe		3
IP Address	4	Modprobe		3
IP Chains Firewall	12		N	
IP Forwarding	11	Netfilter		26
IP Masquerading	10	Network Address Translation Router		9
IP Tables	26	Network File		
Chain	28	DOMAINNAME		11
Command	27	FORWARD_IPV4		11
DNAT	26	GATEWAY		11
Filter	26	GATEWAYDEV		11
Generic Command	27	HOSTNAME		11
Mangle	27	NETWORKING		11
Masquerade	26		P	
Masquerading	26	Private Network		12
NAT	26	Program		
Patch-Criteria	28	diald		12
Rule-Criteria	28	ipchains		12
SNAT	26		R	
Table-name	27	route		
		add -add / netmask		9



add -net	9	U	
add default	9	URL	
Destination	7	clarkconnect.org	17
Flags	8	ipmasq.cjb.net	11
Gateway	7	Userspace	26
Genmask	8	Utility	
Interface	8	ifconfig	7
Metric	8	ipchains	11
References	8	ipfw	11
Use	8	modprobe	3
rules	12	netcfg	6
		route	7
Static Routing	5		