The Linux operating system natively supports packet-filtering rules. Kernel versions 2.2 and earlier support the ipchains command. Beginning with the experimental 2.3 kernel and continuing with the 2.4 kernel, the iptables command is generally used. ipchains and iptables are mutually exclusive; you cannot use both on one system. Nevertheless, both of these commands are used to access the Linux kernel Netfilter feature, which is also supported as a series of modules. Using Netfilter, you can create a

EXERCISE 5.1 (continued)

8. System 1: In the ICMP Types section, select All. In the Action section, select the Drop radio button and then click OK. You will see that you have created an ICMP rule. You must now click the Apply button to make this rule take effect.


10. System 1: Now, create a rule that blocks all traffic from your neighbor’s host to TCP port 80. In the Add Item drop-down box, select the TCP protocol. Under Source, select Host from the Type drop-down box and enter your System 2’s IP address. Under Destination, make sure Any Address is selected, and in the Port drop-down box, select the Equal to (=) option and enter 80 as the port number.

11. In the Action section, select the Drop radio button. Click OK and then click Apply.

Note: The difference between specifying the Deny radio button and specifying the Drop radio button is that Deny causes your host to reply with a UDP packet informing the host that the request was dropped, whereas Drop sends back no message at all.

12. System 2: Use your web browser to test System 1’s HTTP packet-filtering rule.

13. System 1: Create filtering rules for FTP as well, and then have System 2 test these rules.

Using ipchains and iptables

The Linux operating system natively supports packet-filtering rules. Kernel versions 2.2 and earlier support the ipchains command. Beginning with the experimental 2.3 kernel and continuing with the 2.4 kernel, the iptables command is generally used. ipchains and iptables are mutually exclusive; you cannot use both on one system. Nevertheless, both of these commands are used to access the Linux kernel Netfilter feature, which is also supported as a series of modules. Using Netfilter, you can create a
personal firewall for a Linux workstation, or you can create an actual firewall used to create a network perimeter.

All operating systems have a core set of functions, which is called a kernel. In many systems, this core can be upgraded to obtain the latest features and the functionality you need. Some operating systems, such as Linux, allow administrators to compile new kernels. For example, some kernels don’t include the Netfilter function. In such cases, you will have to recompile the kernel. In still other cases, you may have to load modules to augment the kernel. Such modules exist for both ipchains and iptables.

The ipchains and iptables commands have similar syntax, but Netfilter contains several features that require additions. Using either of these commands, you can create packet-filtering rules that accept, drop, or masquerade traffic.

These commands allow you to control packets by manipulating chains, which are specially defined areas of the packet filter designed to hold different rules. Let’s take a look at the elements manipulated in both commands.

Creating Rules

Regardless of whether you are using ipchains or iptables, you must first specify the protocol you wish to block. Because ipchains and iptables are packet-filtering applications, the protocols you specify are limited to ICMP, TCP, and UDP. You can specify individual ports, as well as port ranges. Next, you need to specify a target. ipchains and iptables use similar target names, such as ACCEPT and DENY. Still, each application does use slightly different terminology.

Usually, ipchains and iptables are not set by default. However, recent versions of Red Hat Linux (versions 7.1 and higher), for example, allow you to choose various “firewall” levels during system installation.

Now let’s discuss how you can use both ipchains and iptables to control packets on your host. Study the differences and similarities carefully; you will likely encounter systems that use one or the other.
The `ipchains` application uses three built-in chains, often called *special chains*. They are as follows:

- **input**: Used to control packets entering the interface.
- **output**: Used to control packets leaving the interface.
- **forward**: Used to control packets being masqueraded, or sent to remote hosts.

You must specify a target using the `-j` option. Allowed target built-in values are `ACCEPT`, `DENY`, `REJECT`, `MASQUERADE`, `REDIRECT`, and `RETURN`. The `MASQUERADE` target allows you to establish NAT on a firewall. Case is important for both the chains and the targets. In `ipchains`, all chains are in lowercase letters, and all targets are in uppercase. It is possible to create custom chains that can be used as targets. Usually, built-in chains are adequate.

### Examples of `ipchains`

Suppose, for example, that you have a host with the IP address of 192.168.2.0/24 and that you want to create a simple personal firewall that blocks all incoming ICMP traffic sent from remote hosts to your own host. To do so, you issue the following command:

```
ipchains -A input -p icmp -s 0/0 -d 0/0 -j REJECT
```

This command tells the input chain to forbid any ICMP traffic from any host. If you want to block ICMP traffic from only, say, the 10.100.100.0/24 network, you flush the above rule and replace it with one that specifies only that subnet. The commands to do so are as follows:

```
ipchains -F
ipchains -A input -p icmp -s 10.100.100.0/24 -d 0/0 -j REJECT
```

The host can no longer receive packets, but it can still send them. This is because you have only blocked the input chain. To prohibit this host from sending packets to the 10.100.100.0/24 network, you use the following command to add an entry to the output chain:

```
ipchains -A output -p icmp -s 192.168.2.0/24 -d 0/0 10.100.100.0/24 -j REJECT
```

---

Copyright ©2002 SYBEX, Inc., Alameda, CA www.sybex.com
Now, this host can no longer receive or send ICMP traffic. You are not, of course, limited to controlling just ICMP traffic. If you want to block incoming POP3 traffic from all hosts, you issue the following command:

```
ipchains -A input -p tcp -s 0/0 -d 0/0 110 -j REJECT
```

If you want to deny all traffic by default and then specifically allow only, say, POP3 traffic, you could use the `-P` option, which sets a policy for the chain you specify. You could then begin to allow the POP3 traffic, as well the DNS service and the ephemeral ports necessary for your system to connect to a POP3 server:

```
ipchains -P output DENY
nipchains -P forward DENY
nipchains -P input DENY
nipchains -A input -p tcp -s 0/0 -d 0/0 110 -j ACCEPT
nipchains -A input -p tcp -s 0/0 -d 0/0 1024: -j ACCEPT
nipchains -A input -p udp -s 0/0 -d 0/0 1024: -j ACCEPT
nipchains -A output -p tcp -s 0/0 -d 0/0 1024: -j ACCEPT
nipchains -A output -p udp -s 0/0 -d 0/0 1024: -j ACCEPT
```

Notice that the last rule allows the system to generate a packet to any host on port 53. This rule allows the use of any DNS server. You could be more specific, if you knew the IP address of your DNS server.

You don’t have to create a full masquerading firewall to understand how `ipchains` and `iptables` work. These examples should be enough to get you started creating firewalls. Still, another short example may be helpful. Suppose that you have an internal NIC (named `eth0`), with the IP address of 192.168.2.1/24, and an external NIC (named `eth1`), with the IP address of 45.9.2.23/24. The following entry would enable all systems that are using the internal NIC as a default gateway to use the Internet:

```
ipchains -A forward -i eth0 -s 192.168.2.0/24 -d 0/0 -j MASQ
```

The above entry adds an entry to the forward chain, which is designed to allow masquerading. The `-i` option specifies the `eth0` interface, which is the internal interface. The `-j ACCEPT` target means that this interface will accept masquerading for the 192.168.2.0/24 network. You can then begin to deny or accept traffic as you see fit.
Linux names the first NIC in a system eth0. The second NIC is eth1, the third eth2, and so forth.

Before you can masquerade a connection, you must enable IP forwarding and IP defragmentation on the system, whether you are using ipchains or iptables. On a Linux system, you do this by issuing the following commands:

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

You can do this manually, or you can enter the commands at the bottom of the /etc/rc.d/rc.local file if you want these settings to be made automatically. To learn more about how to use ipchains to create a full-blown firewall, consult the IPCHAINS-HOWTO at www.linuxdoc.org.

Using iptables

The iptables command is used to manipulate Netfilter. The CIW Security Professional exam does not delve into all of Netfilter’s functionality. However, you will be required to understand some of the basic syntax, much like that in ipchains. Some significant differences exist between ipchains and iptables. First, in iptables, all built-in chains are in uppercase letters. Second, iptables has several different tables, each of which contains various chains. ipchains has only one table—filter. iptables keeps the filter table, then adds two more (which explains why the command name is now iptables instead of ipchains):

- **filter** Contains the INPUT, OUTPUT, and FORWARD chains. This is the default table, and it will report its contents when you list chains using the `iptables -L` command.

- **nat** Used for creating NAT tables. Contains the PREROUTING, OUTPUT, and POSTROUTING tables. The PREROUTING table alters packets as soon as they enter (used when masquerading connections), the OUTPUT table alters locally generated packets, and POSTROUTING alters packets before they are about to be sent on the network. If you are using a Linux kernel that supports modules, you must have the `iptables_nat` module installed to use this table.
mangle Alters the packets. Generally, you do not use this for establishing NAT. This table has two chains: PREROUTING (which alters packets that have entered the system) and OUTPUT (which is used for altering packets that have been generated by the local operating system). If you are using a Linux kernel that supports modules, you must have the iptables_mangle module installed to use this table.

---

NOTE
You can read the filter, nat, and mangle entries by using the `iptables -t` command. Do not uppercase these entries. For example, to list the nat table, you would use the following command: `iptables -t nat -L`. The `-L` option allows you to list various tables and chains.

Allowed target values in `iptables` are DROP, ACCEPT, QUEUE, and RETURN. `iptables` also allows the creation of user-defined chains.

Examples of `iptables`
Because `iptables` has three tables to read instead of one, you need to know how to list and manipulate them. The filter table is listed by default. You use these two commands to list the nat and mangle tables:

```
iptables -t nat -L
iptables -t mangle -L
```

When creating a personal firewall, however, you do not have to use the nat or mangle tables. To create a simple personal firewall that blocks all incoming ICMP traffic, you issue the following command:

```
iptables -A INPUT -p icmp -s 0/0 -d 0/0 -j DROP
```

To block ICMP traffic from only the 10.100.100.0/24 network, you issue this command:

```
iptables -A INPUT -p icmp -s 10.100.100.0/24
```  

To deny all but POP3 traffic, you issue the following commands, after flushing any existing rules:

```
iptables -P INPUT DROP
iptables -P FORWARD DROP
iptables -P OUTPUT DROP
iptables -A INPUT -p tcp -s 0/0 -d 0/0
    --dport 1024: -j ACCEPT
```
Chapter 5 • Firewalls Roles and Types

The first three entries automatically configure the personal firewall to drop any and all connections. Lines 4 through 6 then allow any server on the Internet to connect to your workstation’s ephemeral ports (i.e., ports 1024 and higher). If you know the IP addresses of your DNS and e-mail servers, you could restrict all of the entries to a specific IP address, rather than the entire Internet. If, for example, the IP address of the DNS server was 10.100.100.100/8 and the e-mail server were at the address of 203.54.23.3/24, you would enter the following:

```
iptables -A INPUT -p tcp -s 10.100.100.100/8 -d 0/0 --dport 1024: -j ACCEPT
iptables -A INPUT -p udp -s 10.100.100.100/8 -d 0/0 --dport 1024: -j ACCEPT
iptables -A OUTPUT -p udp -s 0/0 -d 203.54.23.3/24 --dport 53 -j ACCEPT
iptables -A OUTPUT -p tcp -s 0/0 -d 203.54.23.3/24 --dport 110 -j ACCEPT
```

Make sure that you flush any existing rules. An old rule that you may have forgotten about could be causing you a problem.

If you want to masquerade a connection using `iptables`, you would use the `nat` table. Using the same scenario as the `ipchains` command, you would masquerade your internal network so that it could connect to the Internet as follows:

```
iptables -t nat -A POSTROUTING -o eth1 -j MASQUERADE
```

For more information on using `iptables`, read the IPTABLES-HOWTO at http://netfilter.samba.org/unreliable-guides/packet-filtering-HOWTO/packet-filtering-HOWTO.linuxdoc.html and other locations on the Internet. If you want to learn more about masquerading using a Linux system, consult the following URL:

www.linuxdoc.org/HOWTO/IP-Masquerade-HOWTO.html
Using ipchains and iptables

Logging with **iptables** and **ipchains**

Both the filter and mangle tables contain additional chains. The `iptables` command also allows additional logging options. For example, the `ipchains -l` option causes a rule to log any match and send a message to the `/var/log/messages` file. The `iptables` command, however, requires that you use the `-j` option and specify the target `LOG` for any rule. You can learn more about `ipchains` and `iptables` by consulting their respective man pages.

Switching a Linux System between **ipchains** and **iptables**

Because most Linux systems use modules to extend the functionality of the kernel, use the `lsmod` command to verify what modules are installed. Look for entries such as `ipchains`, `ip_tables`, or `iptables_filter`. If you have kernel 2.4 and later but the `ipchains` module is installed, issue the following command:

```
modprobe -r ipchains
```

This command removes all `ipchains` modules. You can then load all of the `iptables` modules, if they are present:

```
modprobe ip_tables
```

In many systems, simply issuing the `ipchains` or `iptables` command will automatically load the necessary modules. If the `iptables` modules are not present, then install them from `www.rpmfind.net`. If these modules will not install, you need to recompile your kernel to use Netfilter. Most systems, however, ship with kernels that support Netfilter.
To remove iptables, you can issue the following commands:

- `modprobe -r iptables`
- `modprobe -r iptable_filter`

You can then use the `modprobe ipchains` command to reinstall ipchains, if you wish. It is wise to use `lsmod` often to determine what else you need to install or uninstall. Remember that some systems use monolithic kernels, which means that they will not allow the use of modules. In such cases, you will have to recompile the kernel to include Netfilter, or to allow modules.

**EXERCISE 5.2**

**Using the `ipchains` Command to Create a Personal Firewall in Linux (for Kernels 2.2 and Lower, or for Systems Running the `ipchains` Module)**

In this exercise, you will use the `ipchains` command to create packet-filtering rules for your system. This exercise implies the use of two systems, which are helpful for testing purposes.

Note: If you have a kernel of 2.2 or earlier, `ipchains` is generally used. Use the `uname -a` command to verify your kernel version before continuing with this exercise. Also, use the `/sbin/lsmod` command to determine which module is currently installed.

1. Boot into Linux and log on as root. Verify that each of your systems can connect with each other.

2. System 1: Verify that the `ipchains` module and command are installed:

```
host# rpm -qa | grep ipch
ipchains-1.3.9-1
```
If the ipchains RPM is not installed, obtain it from www.rpmfind.net.

3. System 1: To block ICMP to all hosts, issue the following command:

    ipchains -I input -i eth0 -p icmp -s 0/0 -d 0/0 -l -j REJECT

This command causes ipchains to add a rule to the input chain that rejects all ICMP packets from all sources (-s 0/0) to all destinations (-d 0/0). The -i option specifies an interface (e.g., eth0, eth1, eth2, and so forth). The -l option sends a log entry to the /var/log/messages file for every ICMP packet received.

4. System 2: Ping your system. You will not receive any echo reply messages, and you will see error messages. Do not press Ctrl+C to stop pinging the host; just let the error messages continue on your screen.

5. System 1: List all your chains:

    host# ipchains --line-numbers -nL

You will see that the input chain contains the ICMP rule you created, complete with a line number. The -n option disables DNS lookups, which is helpful in case no DNS resolution is available. Otherwise, the listing will “hang” indefinitely waiting for resolution to occur. This option is helpful when you have a long list of rules.

6. System 1: When System 2 is ready, issue the following command to flush all rules from the input chain:

    host# ipchains -F

7. System 2: Notice that when you flush the rules for this system, it is possible to ping it. Ping the system to verify that you have flushed your ipchains rules.

8. System 1: Verify that your web server is running, and verify that it is serving web pages. To block all traffic to your web server, issue the following command:

    ipchains -I input -i eth0 -p tcp -s 0/0 -d 0/0 80 -l -j REJECT
9. System 2: Use a web browser, such as Lynx, to access your web server.

10. System 1: Add the following rule for FTP traffic:
    
    \[ \text{ipchains} \ -I \text{ input} \ -i \text{ eth0} \ -p \text{ tcp} \ -s 0/0 \ -d 0/0 \ 21 -l -j \text{ REJECT} \]

11. System 1: Issue the following command to view your kernel output file:
    
    \[ \text{host}\# \text{ tail} \ -f \ /var/log/messages \]

12. System 2: Use an FTP client to test this rule.

13. System 1: You will see information concerning each packet that is received. Test whether your system can log ICMP and TCP packets sent to port 80.

14. System 1: List the rules that you have established. Now, delete the FTP rule. If, for example, the FTP rule is the third rule, you would issue the following command:
    
    \[ \text{host}\# \text{ ipchains} \ -D \text{ input} \ 3 \]

15. System 1: Before you flush all these rules, use the following command to save your existing rules to a text file:
    
    \[ \text{host}\# \text{ ipchains-save} \ > \text{iptablesrules.txt} \]

16. Both systems: When you have finished, use \text{ipchains} \ -F to erase all chains. Failure to perform this step will cause problems in the future.

**EXERCISE 5.3**

Using the \textit{iptables} Command to Create a Personal Firewall in Linux (for Kernels 2.3 and Higher)

In this exercise, you will use the \texttt{iptables} command to create packet-filtering rules for your system. Although you do not have two NICs, you can create a personal firewall for your system.
EXERCISE 5.3 (continued)

Note: Systems using kernels 2.3 or later often use iptables. Use the `uname -a` command to verify your kernel version before continuing with this exercise. Also, use the `/sbin/lsmod` command to determine which module is currently installed.

1. Boot into Linux and log on as root. Verify that this system can connect with others on the network.

2. Verify that the iptables module and command are installed:

   ```
   host# rpm -qa | grep iptab
   ipchains-1.2.1a-1
   ```

   If the iptables RPM is not installed, obtain it from `www.rpmfind.net`. Make sure that you are using kernel 2.3 or higher.

3. To block ICMP to all hosts, issue the following command:

   ```
   iptables -I INPUT -i eth0 -p icmp -s 0/0 -d 0/0 -j DROP
   ```

   Note: This command causes iptables to add a rule to the INPUT chain that rejects all ICMP packets from all sources (-s 0/0) to all destinations (-d 0/0). In iptables, the `-i` option specifies the input device. Notice that you must specify the INPUT chain, not the input chain; case is important.

4. Now, log this rejection:

   ```
   host# iptables -I INPUT -i eth0 -p icmp -s 0/0 -d 0/0 -j LOG
   ```

5. Go to a separate system and ping the first system. You will not receive any echo reply messages, and you will see error messages. Do not press Ctrl+C to stop pinging the host; just let the error messages continue on your screen.

6. List all your chains on both systems:

   ```
   host# iptables --line-numbers -nL
   ```

   You will see that the INPUT chain contains the ICMP rule you created, complete with a line number. This option is helpful when you have a long list of rules.
Exercise 5.3 (continued)

7. On the separate system, issue the following command to flush all rules from the INPUT chain:

```bash
host# iptables -F
```

8. After flushing the chains on the second system, you will be able to ping your host again. Do this, now, to verify that you have flushed all `iptables` tables and chains.

9. Verify that your web server is running on the first system, and verify that it is serving web pages using your second system. Then, on the second server, block and log all traffic to your web server using the following commands:

```bash
iptables -I INPUT -i eth0 -p tcp -s 0/0 -d 0/0 --dport 80 -j DROP
iptables -I INPUT -i eth0 -p tcp -s 0/0 -d 0/0 --dport 80 -j LOG
```

10. Use a web browser, such as Lynx, to access the second web server.

11. On any system, add the following rule for FTP traffic:

```bash
iptables -I INPUT -i eth0 -p tcp -s 0/0 -d 0/0 --dport 21 -j DROP
iptables -I INPUT -i eth0 -p tcp -s 0/0 -d 0/0 --dport 21 -j LOG
```

12. After adding the above rule, issue the following command to view your kernel output file:

```bash
host# tail -f /var/log/messages
```

13. Now, use an FTP client to test this rule.

14. You will see information concerning each packet that is received. Work with the remote system to test whether your system can log ICMP and TCP packets sent to port 80.

15. List all of the rules that you have established. Now, delete the FTP rule. If, for example, the FTP rule is the third rule, you would issue the following command:

```bash
host# iptables -D INPUT 3
```