Chapter 1: Configuring Linux

In This Chapter

- **Removing unused services**
- **Installing a firewall and IP security**
- **Using logging**

To make your Linux system secure enough to resist being compromised, you should focus on two general areas:

- ✦ The Linux system itself, which is the focus of this chapter
- ✦ The security of the network to which the Linux system is connected

You need to secure the Linux system before attaching it to the network. It is difficult to ensure the security of a system if it has been connected to a public network before you begin configuring it for secure operation. Because this chapter focuses on prevention, you need to begin with a clean slate — in other words, a system that you are certain has not been turned into a hacker's willing pawn.

If your Linux system will be exposed to the Internet, you want to protect it from as much foul play as possible. You also want to protect your private network resources from the Internet and the Linux system allowing public access. To do this, you need to design a detailed security plan and develop what is commonly called an isolation LAN or demilitarized zone (DMZ) on your network. The simplest form of a secure LAN consists of two routers, as shown in Figure 1-1.

The *outside* router controls access to systems within the secure LAN and restricts (generally blocks entirely) connections that originate from hosts on the internet that are targeting either the private network or the inside router. The *inside* router is designed to allow private LAN traffic to pass outside the network and act as a second barrier between the private network and the internet. It blocks all inbound connections and specifically blocks connections initiated by hosts on the secure LAN and the outside router itself. This ensures that if a secure LAN host or the outside router is compromised, intruders will not have gained access to the private network. Even with a secure LAN configuration, your Linux host needs to be securely configured. Although the secure LAN goes a long way towards protecting your private network resources, your hosts on the secure LAN are exposed to the Internet.

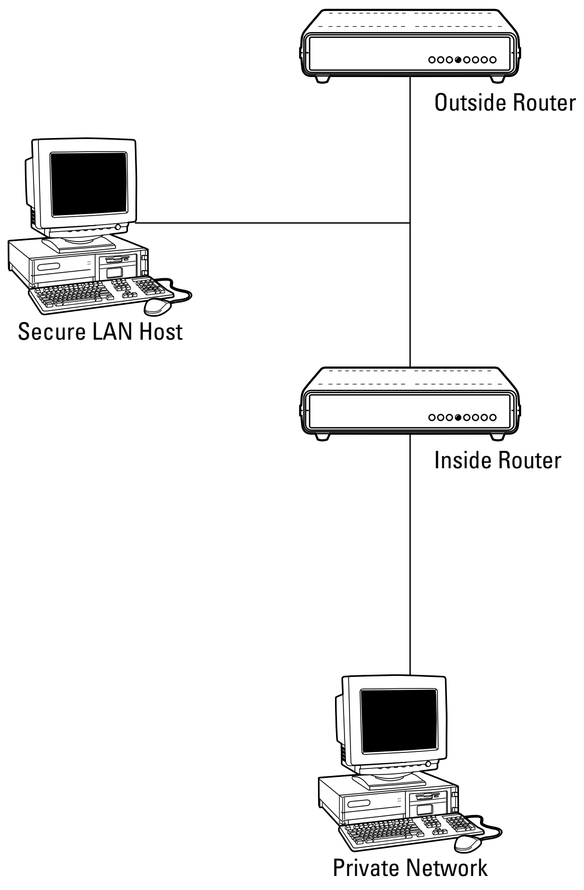


Figure 1-1: Secure LAN configu-ration.

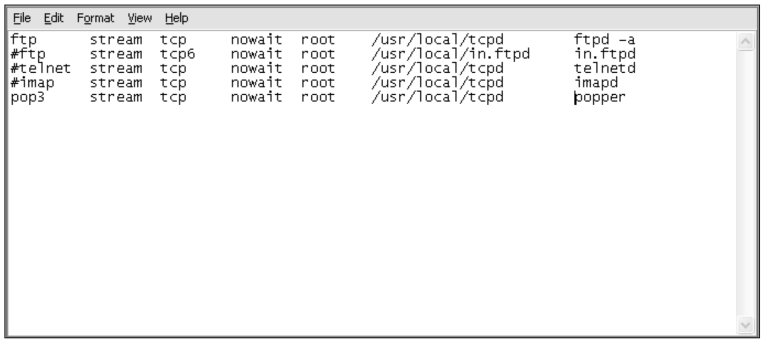
Getting a Handle on TCP and UDP Services

When you finish your Linux installation and before installing any other software, you need to remove the default network services that won't be used. By default, the services running on your new Linux installation are providing what the install package's creator deems commonly used services. Because your system will provide Web services (exclusively if possible) and because it may be directly connected to the Internet, you will want to remove many of the default services. Any service can provide an unscrupulous remote user with a potential avenue into your system. If the service is available to network users and is running, you should consider it a potential weakness. Even though many hours are spent testing and identifying flaws in the Linux system, testing can't catch everything. However, you can minimize potential threats by removing unneeded services.

Inetd

Identifying and controlling services, network services in particular, is fairly straightforward on most current versions of Linux, including Red Hat, Mandrake, and SuSE. Within the /etc directory, you will find a file called inetd.conf. This file is used at system startup to load most of the system's networking services. Rather than have a large number of active processes listening for network connections, the inetd "super server" plays traffic cop between service requests and available services, which results in a more efficient use of system resources. Basically, inetd is a server service that listens for requests that relate to TCP and UDP connections. Any requests for services are then passed to the appropriate service if one is available. If no service is available, the request is dropped. For example, if a Linux server is allowing Web and FTP connections for hundreds of users, inetd allows the creation of sessions for each individual connection rather than have numerous instances of the ftpd or httpd processes idling all the time waiting for users to connect. This frees up memory and CPU time for other uses.

The inetd process relies on a corresponding conf file to determine which processes are available for use. Figure 1-2 shows an example of an inetd.conf file.



```
ftp      stream  tcp    nowait  root    /usr/local/tcpd        ftpd -a
#ftp     stream  tcp6   nowait  root    /usr/local/in.ftpd     in.ftpd
#telnet  stream  tcp    nowait  root    /usr/local/tcpd        telnetd
#imap    stream  tcp    nowait  root    /usr/local/tcpd        imapd
pop3     stream  tcp    nowait  root    /usr/local/tcpd        popper
```

Figure 1-2: Sample inetd. conf.

What you find in the inetd file varies depending on the distribution you use, but how you can control it is the same across most distributions. You can open the file with the text editor of your choice and add or remove entries as needed. For example, the inetd server ignores any line with the # symbol. Using the # symbol is also known as "commenting out" the code that follows.

"Commenting out" allows you to disable startup services without removing the configuration. This is useful if you have a service you will use only occasionally and you want to start it manually instead of at startup. You can also temporarily disable one of the services by inserting the # symbol and restarting the inetd. As the phrase "comment out" suggests, you can also use this symbol to insert comments into the file. For example you might want to note why you added or removed a particular entry. The process for restarting inetd varies in some distributions, but in general, you can go to the directory where the inetd is stored (`/etc/rc.d/inet.d` on Red Hat) and type the `inetd restart` command. You need to run this command as root or with root privileges. If this command doesn't work on your distribution, search that product's support information for the relevant process to restart the inetd process.

The `inetd.conf` file contains six attributes for each item. Here are the attributes from left to right as shown in Figure 1-2:

✦ **The service name:** In the example, the first line has a service named ftp, indicating it is an ftp service. The name of the service is taken from the corresponding entry in the `/etc/services` file.

✦ **The socket type:** The socket type can be either stream or datagram. In most cases, it's stream.

✦ **The protocol type:** This can be specified as either TCP or UDP.

✦ **The wait/nowait option:** This option determines if a new process is spawned for each request (nowait) or if each request is processed sequentially one at a time (wait).

✦ **The user whose rights are used to execute the service:** The user is root in the example.

✦ **The path to the command that launches the daemon:** The path is `/usr/local/tcpd` in the example. A *daemon* is a program that waits in the background until called upon by a user to actually do something.

The seventh field is optional. If you don't need to specify any execution arguments, feel free to omit this field.

TIP

Many Linux distributions include graphical tools for configuring the startup services. Typically these configuration applets contain a list of startup services and corresponding check boxes. To disable a service, you simply clear the corresponding check box. In most cases, you need to manually shut down the service by using a command in a terminal window or by restarting the system. For more information about the specific tools available, consult the developers of your desktop environment (KDE or Gnome).

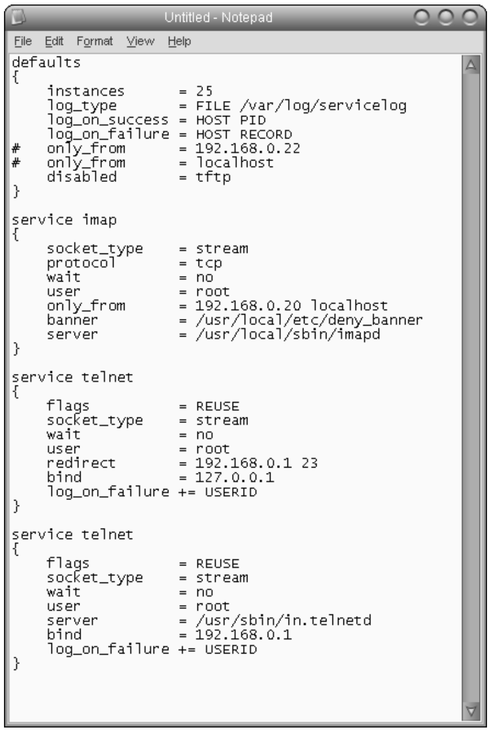
Xinetd

On many current distributions, the xinetd daemon is used instead of inetd. Conceptually, xinetd does the same thing inetd does. However, for the most part, this similarity is only skin deep. The processes for configuring files and for the construction of the configuration files are quite different. There are some other critical operational differences. For example, any user can use xinetd to start servers that do not need privileged ports. This is possible because, unlike inetd, xinetd does not require that the services in its configuration file (`xinet.d`) also appear in `/etc/services`.

In the list below, the key enhancements (what xinetd does that inetd does not) are outlined.

- ✦ Access control works on multi-threaded and single-threaded services.
- ✦ Xinetd kills those services that no longer meet the existing access control criteria and those that have been removed from the `xinet.d` configuration file.
- ✦ It can place limits on the number of processes a single host can spawn, the number of servers a particular service can start, and the size of log files. These configurable limitations go a long way towards preventing denial-of-service attacks, which attempt to disrupt the system by overutilizing available resources.

As noted earlier, the configuration file (`/etc/xinetd.conf`) for the xinetd service is also notably different than the file used by inetd. The example below (Figure 1-3) shows telnet, IMAP, and the default entries. By itself, this configuration is not entirely useful; I've included it because it illustrates the differences between the layout of inetd and xinetd.



```
defaults
{
    instances      = 25
    log_type       = FILE /var/log/servicelog
    log_on_success = HOST PID
    log_on_failure = HOST RECORD
#   only_from      = 192.168.0.22
#   only_from      = localhost
    disabled       = tftp
}

service imap
{
    socket_type    = stream
    protocol       = tcp
    wait           = no
    user           = root
    only_from      = 192.168.0.20 localhost
    banner         = /usr/local/etc/deny_banner
    server         = /usr/local/sbin/imapd
}

service telnet
{
    flags          = REUSE
    socket_type    = stream
    wait           = no
    user           = root
    redirect       = 192.168.0.1 23
    bind           = 127.0.0.1
    log_on_failure += USERID
}

service telnet
{
    flags          = REUSE
    socket_type    = stream
    wait           = no
    user           = root
    server         = /usr/sbin/in.telnetd
    bind           = 192.168.0.1
    log_on_failure += USERID
}
```

Figure 1-3: Example xinetd configuration file.

TIP

If you would like to convert an inet.d configuration file to be compatible with xinetd, you can make use of one of a couple of conversion tools. Both itox and xconv.pl come with full distributions of xinetd and are relatively painless to use. For guidance, check out the information located here (under the configuration section): www.linuxfocus.org/English/November2000/ article175.shtml.

No Sharing: NFS/RPC

Unless you're dying for someone to turn your Linux server into the technological equivalent of Typhoid Mary, avoid the Remote Procedure Call (RPC) and Network File Service (NFS). Keep in mind I'm talking about a Linux server exposed to the Internet. RPC and NFS can be very useful on a private network but have no business on at-risk systems. Both services are designed to allow remote users to access resources:

✦ RPC allows the sharing of processing resources among computers. With RPC, a single program can use the resources of several computers collectively in a clustery arrangement.

✦ NFS is used to facilitate disk access. Remote users who have the appropriate access rights can mount and access data on an NFS-enabled hard disk.

It is painfully easy to misconfigure these services, NFS in particular, in a manner that creates huge holes. For example, a misconfigured NFS implementation allows anyone to mount and access data on the NFS-capable drive.

The portmapper controls the use of NFS and RPC. The portmapper is another daemon and is typically housed at /etc/rc.d/rc.inet2, though the location can vary depending on the distribution. If you do not know the location, you can try running the find command on the /etc directory. For example, the following line finds instances of words containing *map:*

```
find /etc -name "*map*" -print | more
```

You can replace the *map* argument with *nfs*, *port*, or *rpc* to thoroughly scan the /etc/rc.d directories for the startup locations of the NFS and RPC services. Like other configuration files, find the relevant entries and place a # in front of the execution lines or simply delete the entries. After the services have been disabled, you need to restart the system to unload them.

REMEMBER

If you want to access your server remotely, Telnet is not a good option because it has no system for protecting information passed between the connecting client and the server. Instead, you should use the Secure Shell (SSH) client-server utility. SSH encrypts all the data that passes between the client and the server. Not only is the logon information protected, but the data transferred is protected as well. Virtually all modern Linux distributions include an SSH implementation, but if it is not present, you can download it. To install SSH, you can download the source, run the `./configure` script, and then run the `make` and `make install` commands. For more information on the installation process for SSH, see `www.openssh.com`.

Misleading Binaries (SUID/GUID Binaries)

In addition to network services, some of the programs on your Linux system can be real liabilities because they need root privileges to do much. Some programs use a specialized bit (set on the binary itself) — called either Set User ID (SUID) or Group User ID (GUID) — to gain more privileges than the user who might run the program or use its services. For example, if users want to send e-mail, they need to use the services of an SMTP server such as Sendmail. However, they probably won't have all the rights needed to perform all the tasks that the mail server needs to do. To perform the needed tasks, Sendmail often runs with the Set User ID bit set to use root account privileges.

Any program that uses SUID/GUID to run with elevated privileges is a liability. If a malicious user gains control of the application, that user may be able to interact with the system using the elevated system rights provided by the SUID/GUID setting. Fortunately, you can easily identify the binaries that have been configured to use the SUID/GUID funtionality. If you check the file attributes of a binary and see an `s` where you would normally see an `r` or `x` (`r-sr-xr-x`, for example), you have a binary using SUID/GUID. So how do you find and, if not needed, disable these binaries? The easiest method is to use the `find` command to search the entire system and build a list of the SUID/GUID-enabled applications. Here's an elegant solution that uses the `find` command with the option to dump the output to a text file:

```
find / \( -perm -4000 -o -perm -2000 \) -exec ls -ldb {} \;
>> /tmp/suids
```

This command creates the text file `suids` in the `/tmp` directory with information about all the binaries using the SUID/GUID capability. You can find and delete any binaries that you know you don't need. If you determine that some of the binaries may prove useful, you can refine their access rules (by using the `CHMOD` command) so that not just anyone can run them. This is especially useful if one of the remaining binaries needs to be granted root access.

Here are a couple handy ways to help you determine if you need a particular service or program:

✦ Visit the home page of your Linux distribution and search for the program.

✦ Search for "man *program/service*" using a search engine like Yahoo! or Google. The manual (man) pages typically reveal a wealth of information about the program/service and its usage.

Protection with Firewalls

Although many third-party security products, including firewalls, are available for Linux, several handy features are included with the operating system out of the box. Having a firewall on your Linux system is exceptionally important if the system is outside any other firewalls. For example, if the resources for setting up a secure LAN are not available, the Linux server you build needs a local firewall installed if it is connected to the Internet. Even if you're using a secure LAN configuration, an additional, local firewall adds an extra line of defense to stave off intruders if they compromise one of your existing routers.

Ipchains

Ipchains is a long-standing firewalling mechanism that is available on most Linux distributions that use kernel version 2.2.x. To get started, you need to enable ipchains. If your system is using the IP address 10.50.4.55 with the mask 255.255.255.0 (internally), you need to run the following commands with root privileges:

```
echo 1 > /proc/sys/net/ipv4/ip_forwardipchains -A forward -j
MASQ -s 10.50.4.0/24 -d 0.0.0.0/0
```

The first command enables `ip_forwarding`, and the second entry configures the server with which network to grant Internet access. If you want to load the ipchains service at startup, you need to place the two preceding commands into the `etc/rc.d/rc.local` file. After you have the firewall running, you can use the `ipchains -L` command to see the configured IP filtering rules. Of course to start with, there is not much to see. The output of this command looks something like Figure 1-4.

In the figure, three chains are in place by default. The chains are for inbound, outbound, and forwarded packets. But because you have not configured the behavior of these chains, nothing useful is happening at this point. To set a rule, you need to invoke the `ipchains` command and tell it what to do. Take a look at the following example:

```
ipchains -A input -p icmp -i eth0 -j REJECT
```

Here's a closer look at the components of this example:

✦ The `ipchains -A input` portion indicates that you're appending (`A`) a new rule to the ipchains `input` chain. In place of `-A`, you can use `-L` to see the current ipchains rules and `-D` to delete a rule from a chain.

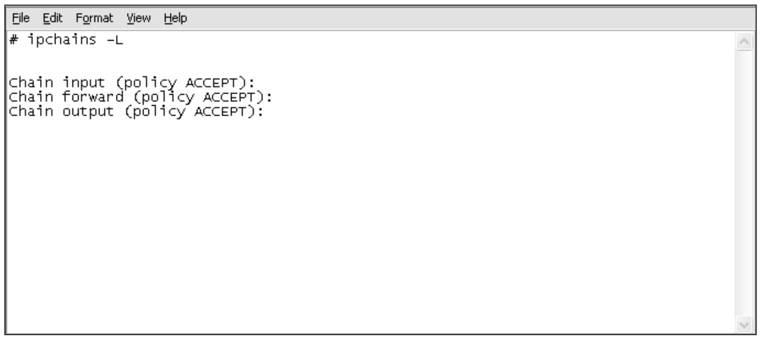
✦ The option `-p` specifies that you're going to set a rule for a protocol.

✦ Net is the type (used with the `-p` option), and in this case, that is all ICMP traffic.

✦ `Eth0` is the network interface to apply the rule to.

✦ The `-j REJECT` option specifies that the ICMP traffic should be rejected at the interface. The result is that any inbound ICMP traffic on the `Eth0` interface is summarily rejected.

See Book I, Chapter 3, for information about products that make ipchains and other firewall configuration easier. This chapter only touches on the capabilities of the ipchains tool. If you're interested in finding out more about ipchains, visit `www.rt.com/man/ipchains.8.html` or type "man ipchains" on your Linux server to see the almost overwhelming number of ipchains run-time options.



```
# ipchains -L


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

Figure 1-4: Example ipchains rules configuration.

Iptables

Another option for implementing a firewall on your Linux system is through the use of iptables. They are similar in function (they both control network access); however, iptables has several advantages over ipchains. The advantages are noteworthy enough that if you are using a 2.4.x or newer kernel version (any build of Red Hat newer than 7.1, for example) you should use iptables. Below are some of the ways in which iptables differs from ipchains.

✦ Iptables, unlike ipchains, uses a single chain to process filtered packets. Because of this, iptables allows you to exert finer control over the kinds of filtering options you choose. For example, a packet forwarded by an ipchains host must traverse the input, forward, and output chains to make it to its destination. An iptables host must only check the packet against the output filter. If the host is using iptables, a conflicting rule on the input chain won't stop the forwarding of the packet.

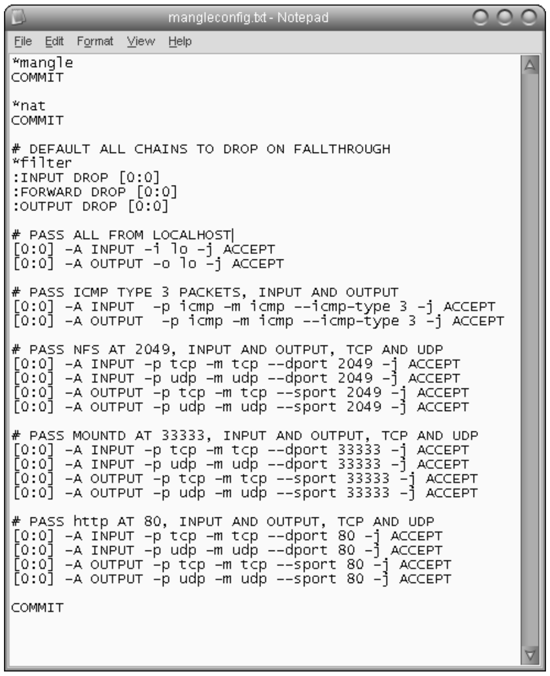
- The iptables host processes inbound packets using only the inbound rules, and outbound packets using only the outbound rules. If you want to specify the network interface to be used with a particular rule, you must use the appropriate option in your configuration. The `-i` option must be used with the inbound or forward chains and the `-o` option must be used with the outbound chains. Remember that the inbound interface will not use outbound rules and the outbound interface will not use inbound rules. If you mix your options, the rule will not have the intended effect.
- Unlike ipchains, you must specify the source or destination port after the protocol declaration (ICMP, TCP, or UDP). In ipchains the order was not all that important, however with iptables the rules will fail to operate as expected if the proper order is not used.

As you might expect, the configuration file (typically located in `/etc/sysconfig/iptables`) has changed considerably as well. The overall arrangement is a bit cleaner and if you use comments, it is fairly straightforward to understand what is going on in the configuration. In Figure 1-5, the configuration first sets a rule that allows the local host (running iptables) to pass any traffic to or from *localhost*. The next entry allows the passage of ICMP type 3 traffic. Type 3 "Destination Unreachable" messages are used when systems that forward data to another host cannot find the targeted host. Next, there are mountd and NFS connections allowed. These would probably be essential if you needed to connect to your system over a network to send data (to upload Web page content for example), but if you are working locally on the machine, these could be eliminated. The mountd entry in this example only works if you go to the NFS configuration file `/etc/rc.d/init.d/nfs` and assign the mountd service to a single port (33333 in this case). By default, the port is picked during system startup and varies between 32000 and 42000. Finally, the configuration file allows the passing of http traffic on port 80. If you need to use https as well, you could duplicate this last entry and change the destination port (dport) and source port (sport) values to 443.

In Figure 1-4, note that three chains are in place by default.

TIP

I strongly recommended that you become familiar with the syntax and use of ipchains and iptables before you use either in a production environment. It's easy to overlook something and leave your system exposed. Make a plan for the kinds of traffic you need to allow in and out of your server and for the services and ports that need to be blocked. With a clear plan, your results will be much better. To find additional information about ipchains configurations, check out `www.linux.org/docs/ldp/howto/IPCHAINS-HOWTO.html`. If you will be working with iptables, there is a great source of general information and configuration examples located at `www.jollycom.ca/iptables-tutorial/iptables-tutorial.html`.



```
*mangle
COMMIT

*nat
COMMIT

# DEFAULT ALL CHAINS TO DROP ON FALLTHROUGH
*filter
:INPUT DROP [0:0]
:FORWARD DROP [0:0]
:OUTPUT DROP [0:0]

# PASS ALL FROM LOCALHOST
[0:0] -A INPUT -i lo -j ACCEPT
[0:0] -A OUTPUT -o lo -j ACCEPT

# PASS ICMP TYPE 3 PACKETS, INPUT AND OUTPUT
[0:0] -A INPUT  -p icmp -m icmp --icmp-type 3 -j ACCEPT
[0:0] -A OUTPUT  -p icmp -m icmp --icmp-type 3 -j ACCEPT

# PASS NFS AT 2049, INPUT AND OUTPUT, TCP AND UDP
[0:0] -A INPUT -p tcp -m tcp --dport 2049 -j ACCEPT
[0:0] -A INPUT -p udp -m udp --dport 2049 -j ACCEPT
[0:0] -A OUTPUT -p tcp -m tcp --sport 2049 -j ACCEPT
[0:0] -A OUTPUT -p udp -m udp --sport 2049 -j ACCEPT

# PASS MOUNTD AT 33333, INPUT AND OUTPUT, TCP AND UDP
[0:0] -A INPUT -p tcp -m tcp --dport 33333 -j ACCEPT
[0:0] -A INPUT -p udp -m udp --dport 33333 -j ACCEPT
[0:0] -A OUTPUT -p tcp -m tcp --sport 33333 -j ACCEPT
[0:0] -A OUTPUT -p udp -m udp --sport 33333 -j ACCEPT

# PASS http AT 80, INPUT AND OUTPUT, TCP AND UDP
[0:0] -A INPUT -p tcp -m tcp --dport 80 -j ACCEPT
[0:0] -A INPUT -p udp -m udp --dport 80 -j ACCEPT
[0:0] -A OUTPUT -p tcp -m tcp --sport 80 -j ACCEPT
[0:0] -A OUTPUT -p udp -m udp --sport 80 -j ACCEPT

COMMIT
```

Figure 1-5: Example iptables configuration.

Playing Games: IP Masquerade

Network Address Translation (NAT) is used to hide a group host address behind a single public address. Typically NAT is used to allow multiple private hosts to use a single public IP address to access the Internet. Everyone on the private network sharing the single public IP is effectively "masked" — hence the moniker *Masquerade* — from the Internet. The individual host can use a private IP address (192.168.1.22, for example) and use NAT to access public networks with a different IP address that is routable on the Internet. The advantage to this configuration is that an organization with many internal hosts needs only a single publicly known IP address for all of its resources.

To use IP Masquerade, you need a build that already supports it, or you may need to recompile your kernel to use IP Masquerade. Most current Linux releases support IP Masquerade. To check your system, you can run this command:

```
uname -a
```

The results reveal the kernel version your system is using. To use IP Masquerade, you need kernel version 2.0.x or greater. On the Red Hat distribution, within the `/etc/rc.d/rc.local` script is an entry that loads the

/etc/rc.d/rc.firewall script during system startup. The purpose of the rc.firewall script is to load all the needed modules to run IP Masquerade and process its relevant configuration files. This file is where you enter the traffic restrictions (rule sets) like the ipchains and iptables examples earlier in this chapter. Many sample IP Masquerade configurations are available, including www.tldp.org/HOWTO/IP-Masquerade-HOWTO/, which helps you get your masquerade configuration up and running.

Keeping on Top with Logging

Although it's important to have a handle on the services running on your system and use firewalling if possible, comprehensive logging is one of the most useful methods for ensuring that your system remains secure. The syslog daemon is used on Linux and Unix systems to facilitate logging. On most Linux installations, a global configuration file controls the behavior of the syslog component. The /etc/syslog.conf file passes the logging instructions to the syslog daemon. Most default Linux installations are designed to balance security and functionality, and as a result, the default logging behavior needs improvement.

Within the syslog.conf file, you will find entries that look like this:

```
*.=info;*.=notice      /usr/log/notice.log
```

Each entry has three elements:

- ✦ The first item is the source of the log information, called a *facility* — in other words, which processes should be monitored for logging output. * is used to specify "all available" log sources.
- ✦ The information following the * specifies the kinds of events that will be captured, known as *priority.* For example, in *.=notice, all log sources are monitored, and only the notice priority events are captured.
- ✦ The last element specifies what will be done with the logging information, and in most cases where the log information will be stored. In the example, the logging output is stored in the file /usr/log/notice.log.

It is fairly useful to log different facilities to different log files. Although this can increase the amount of disk space used by log files, it can make locating information about particular events easier.

Several facilities that you should track may not be logged by default. To begin logging these items, you need to create a new entry in the syslog.conf log file. Table 1-1 shows some of the common facilities that you should use.

Table 1-1 Common Logging Facility Options

Facility	*Use*
`Auth/Security`	Log information relating to user authentication.
`Authpriv`	A more detailed version of auth.
`Cron`	Logs activity of the cron daemon, which is used to schedule events.
`Daemon`	Collects information about *all* system daemons.
`Mail`	Tracks mail server activity. This is a must have if you're running a mail server.
`Syslog`	Information about the syslog server itself.

Along with this selection of facilities, you need to understand the priorities that are available. How much detail you need is up to you. Keep in mind that if a problem occurs and you don't have enough information to identify the source, the logs are essentially useless. However, if you collect every transaction and message in the log, it may be nearly impossible to locate meaningful information. Table 1-2 describes the available priorities.

Table 1-2 Logging Priorities

Priority	*Use*
`Info`	Logs general informational messages
`Notice`	Logs occurrences that require special interaction (that is, user prompts)
`Warning`	System warning, "might be a problem"
`Err`	System error, "a problem occurred"
`Crit`	Critical condition, "a big problem has occurred"
`Alert`	Intervention required, "problem happening, need help"
`Emerg`	Serious system failure, "something is hosed"
`Panic`	System panic, "HUGE problem"
`Debug`	Logs nearly everything

d Bk01Ch01.qxd 11/4/03 10:45 AM Page 22