11

System Administration: Core Concepts

In This Chapter

The job of a system administrator is to keep one or more systems in a useful and convenient state for users. On a Linux system, the administrator and user may both be you, with you and the computer being separated by only a few feet. Or the system administrator may be halfway around the world, supporting a network of systems, with you being simply one of thousands of users. A system administrator can be one person who works part-time taking care of a system and perhaps is also a user of the system. Or the administrator can be several people, all working full-time to keep many systems running.

blank

SECURING A SERVER

You may secure a server either by using TCP wrappers or by setting up a chroot jail.

TCP WRAPPERS: CLIENT/SERVER SECURITY (hosts.allow AND hosts.deny)

When you open a local system to access from remote systems, you must ensure that the following criteria are met:

- Open the local system only to systems you want to allow to access it.
- Allow each remote system to access only the data you want it to access.
- Allow each remote system to access data only in the appropriate manner (readonly, read/write, write only).

As part of the client/server model, TCP wrappers, which can be used for any daemon that is linked against **libwrap.so**, rely on the **/etc/hosts.allow** and **/etc/hosts.deny** files as the basis of a simple access control language. This access control language defines rules that selectively allow clients to access server daemons on a local system based on the client's address and the daemon the client tries to access.

Each line in the **hosts.allow** and **hosts.deny** files has the following format:

daemon_list : client_list [: command]

where ***daemon_list*** is a comma-separated list of one or more server daemons (such as **rpcbind**, **vsftpd**, or **sshd**), ***client_list*** is a comma-separated list of one or more clients (see Table 11-3, "Specifying a client," on page 442), and the optional ***command***

is the command that is executed when a client from ***client_list*** tries to access a server daemon from ***daemon_list***.

When a client requests a connection with a local server, the **hosts.allow** and **hosts.deny** files are consulted in the following manner until a match is found:

1. If the daemon/client pair matches a line in **hosts.allow**, access is granted.
2. If the daemon/client pair matches a line in **hosts.deny**, access is denied.
3. If there is no match in either the **hosts.allow** or **hosts.deny** files, access is granted.

The first match determines whether the client is allowed to access the server. When either **hosts.allow** or **hosts.deny** does not exist, it is as though that file was empty. Although it is not recommended, you can allow access to all daemons for all clients by removing both files.

Examples For a more secure system, put the following line in **hosts.deny** to block all access:

```
$ cat /etc/hosts.deny
...
ALL : ALL : echo '%c tried to connect to %d and was blocked' >> /var/log/tcpwrappers.log
```

This line prevents any client from connecting to any service, unless specifically permitted in **hosts.allow**. When this rule is matched, it adds a line to the file named **/var/log/tcpwrappers.log**. The **%c** expands to client information and the **%d** expands to the name of the daemon the client attempted to connect to.

With the preceding **hosts.deny** file in place, you can include lines in **hosts.allow** that explicitly allow access to certain services and systems. For example, the following **hosts.allow** file allows anyone to connect to the OpenSSH daemon (ssh, scp, sftp) but allows telnet connections only from the same network as the local system and users on the 192.168. subnet:

```
$ cat /etc/hosts.allow
sshd : ALL
in.telnet : LOCAL
in.telnet : 192.168.* 127.0.0.1
...
```

The first line allows connection from any system (ALL) to **sshd**. The second line allows connection from any system in the same domain as the server (LOCAL). The third line matches any system whose IP address starts with **192.168.** as well as the local system.

SETTING UP A chroot JAIL

On early UNIX systems, the root directory was a fixed point in the filesystem. On modern UNIX variants, including Linux, you can define the root directory on a per-process basis. The chroot utility allows you to run a process with a root directory other than **/**.

The root directory appears at the top of the directory hierarchy and has no parent: A process cannot access any files above the root directory (because they do not exist). If, for example, you run a program (process) and specify its root directory as **/home/sam/jail**, the program would have no concept of any files in **/home/sam** or above: **jail** is the program's root directory and is labeled **/** (not **jail**).

By creating an artificial root directory, frequently called a (chroot) jail, you prevent a program from accessing or modifying—possibly maliciously—files outside the directory hierarchy starting at its root. You must set up a chroot jail properly to increase security: If you do not set up the chroot jail correctly, you can actually make it easier for a malicious user to gain access to a system than if there were no chroot jail.

USING chroot

Creating a chroot jail is simple: Working as **root**, give the command **/usr/sbin/chroot** ***directory***. The ***directory*** becomes the root directory and the process attempts to run the default shell. Working with **root** privileges from the **/home/sam** directory, give the following command to set up a chroot jail in the (existing) **/home/sam/jail** directory:

```
# /usr/sbin/chroot /home/sam/jail
/usr/sbin/chroot: cannot run command '/bin/bash': No such file or directory
```

This example sets up a chroot jail, but when it attempts to run the bash shell, the operation fails. Once the jail is set up, the directory that was named **jail** takes on the name of the root directory, **/**, so chroot cannot find the file identified by the pathname **/bin/bash**. In this situation the chroot jail is working but is not useful.

Getting a chroot jail to work the way you want is a bit more complicated. To have the preceding example run bash in a chroot jail, you need to create a **bin** directory in **jail** (**/home/sam/jail/bin**) and copy **/bin/bash** to this directory. Because the bash binary is dynamically linked to shared libraries, you need to copy these libraries into **jail** as well. The libraries go in **lib**.

The next example creates the necessary directories, copies **bash**, uses ldd to display the shared library dependencies of bash, and copies the necessary libraries into **lib**. The **linux-gate.so.1** file is a dynamically shared object (DSO) provided by the kernel to speed system calls; you do not need to copy it to the **lib** directory.

```
$ pwd
/home/sam/jail
$ mkdir bin lib
$ cp /bin/bash bin
$ ldd bin/bash
        linux-gate.so.1 =>  (0x0089c000)
        libtinfo.so.5 => /lib/libtinfo.so.5 (0x00cdb000)
        libdl.so.2 => /lib/libdl.so.2 (0x00b1b000)
        libc.so.6 => /lib/libc.so.6 (0x009cb000)
        /lib/ld-linux.so.2 (0x009ae000)
$ cp /lib/{libtinfo.so.5,libdl.so.2,libc.so.6,ld-linux.so.2} lib
```

Now that everything is set up, you can start the chroot jail again. Although all of the setup can be done by an ordinary user, you have to run chroot as Superuser:

```
$ su
Password:
# /usr/sbin/chroot .
bash-3.2# pwd
/
bash-3.2# ls
bash: ls: command not found
bash-3.2#
```

This time the chroot finds and starts **bash**, which displays its default prompt (**bash-3.2#**). The pwd command works because it is a shell builtin (page 247). However, bash cannot find the ls utility (it is not in the chroot jail). You can copy **/bin/ls** and its libraries into the jail if you want users in the jail to be able to use ls.

To set up a useful chroot jail, first determine which utilities the users of the chroot jail will need. Then copy the appropriate binaries and their libraries into the jail. Alternatively, you can build static copies of the binaries and put them in the jail without installing separate libraries. (The statically linked binaries are considerably larger than their dynamic counterparts. The base system with bash and the core utilities exceeds 50 megabytes.) You can find the source code for most of the common utilities in the **bash** and **coreutils** SRPMS (source rpm) packages.

Whichever technique you choose, you must put a copy of su in the jail. The su command is required to run programs while working as a user other than **root**. Because **root** can break out of a chroot jail, it is imperative that you run a program in the chroot jail as a user other than **root**.

The dynamic version of su distributed by Fedora/RHEL requires PAM and will not work within a jail. You need to build a copy of su from the source to use in a jail. By default, any copy of su you build does not require PAM. Refer to "GNU Configure and Build System" on page 513 for instructions on how to build packages such as **coreutils** (which includes su).

To use su, you must copy the relevant lines from the **/etc/passwd** and **/etc/shadow** files into files with the same names in the **etc** directory inside the jail.

Keeping multiple chroot jails

tip If you plan to deploy multiple chroot jails, it is a good idea to keep a clean copy of the **bin** and **lib** files somewhere other than in one of the active jails.

RUNNING A SERVICE IN A chroot JAIL

Running a shell inside a jail has limited usefulness. In reality, you are more likely to need to run a specific service inside the jail. To run a service inside a jail, you must make sure all files needed by that service are inside the jail. The format of a command to start a service in a chroot jail is

```
# /usr/sbin/chroot jailpath /bin/su user daemonname &
```

where *jailpath* is the pathname of the jail directory, *user* is the username that runs the daemon, and *daemonname* is the path (inside the jail) of the daemon that provides the service.

Some servers are already set up to take advantage of chroot jails. For example, you can set up DNS so that **named** runs in a jail (page 804), and the **vsftpd** FTP server can automatically start chroot jails for clients (page 658).

SECURITY CONSIDERATIONS

Some services need to be run as **root**, but they release their **root** privileges once started (Procmail and **vsftpd** are examples). If you are running such a service, you do not need to put su inside the jail.

A process run as **root** could potentially escape from a chroot jail. For this reason, you should always su to another user before starting a program running inside the jail. Also, be careful about which setuid (page 205) binaries you allow inside a jail—a security hole in one of them could compromise the security of the jail. In addition, make sure the user cannot access executable files that he uploads.

DHCP: CONFIGURES HOSTS

Instead of storing network configuration information in local files on each system, DHCP (Dynamic Host Configuration Protocol) enables client systems to retrieve network configuration information each time they connect to the network. A DHCP server assigns IP addresses from a pool of addresses to clients as needed. Assigned addresses are typically temporary, but need not be.

This technique has several advantages over storing network configuration information in local files:

- A new user can set up an Internet connection without having to deal with IP addresses, netmasks, DNS addresses, and other technical details. An experienced user can set up a connection more quickly.
- DHCP facilitates assignment and management of IP addresses and related network information by centralizing the process on a server. A system administrator can configure new systems, including laptops that connect to the network from different locations, to use DHCP; DHCP then assigns IP addresses only when each system connects to the network. The pool of IP addresses is managed as a group on the DHCP server.
- IP addresses can be used by more than one system, reducing the total number of IP addresses needed. This conservation of addresses is important because the Internet is quickly running out of IPv4 addresses. Although a particular IP address can be used by only one system at a time, many end-user systems require addresses only occasionally, when they connect to the Internet. By reusing IP addresses, DHCP lengthens the life of the IPv4 protocol. DHCP applies to IPv4 only, as IPv6 forces systems to configure their IP addresses automatically (called autoconfiguration) when they connect to a network (page 373).

DHCP is particularly useful for administrators who are responsible for maintaining a large number of systems because individual systems no longer need to store unique configuration information. With DHCP, the administrator can set up a master system and deploy new systems with a copy of the master's hard disk. In educational establishments and other open access facilities, the hard disk image may be stored on a shared drive, with each workstation automatically restoring itself to pristine condition at the end of each day.

MORE INFORMATION

Web www.dhcp.org

FAQ www.dhcp-handbook.com/dhcp_faq.html

HOWTO *DHCP Mini HOWTO*

HOW DHCP WORKS

The client daemon, **dhclient** (part of the **dhcp** package), contacts the server daemon, **dhcpd**, to obtain the IP address, netmask, broadcast address, nameserver address, and other networking parameters. The server provides a *lease* on the IP address to the client. The client can request the specific terms of the lease, including its duration; the server can, in turn, limit these terms. While connected to the network, a client typically requests extensions of its lease as necessary so its IP address remains the same. The lease can expire once the client is disconnected from the network, with the server giving the client a new IP address when it requests a new lease. You can also set up a DHCP server to provide static IP addresses for specific clients (refer to "Static Versus Dynamic IP Addresses" on page 368).

DHCP is broadcast based, so both client and server must be on the same subnet (page 371).

DHCP CLIENT

A DHCP client requests network configuration parameters from the DHCP server and uses those parameters to configure its network interface.

PREREQUISITES

Install the following package:

- **dhclient**

dhclient: THE DHCP CLIENT

When a DHCP client system connects to the network, dhclient requests a lease from the DHCP server and configures the client's network interface(s). Once a DHCP client has requested and established a lease, it stores information about the lease in a file named **dhclient.leases**, which is stored in the **/var/lib/dhclient** directory. This information is used to reestablish a lease when either the server or the client needs to reboot. The DHCP client configuration file, **/etc/dhclient.conf**, is required only for custom configurations. The following **dhclient.conf** file specifies a single interface, **eth0**:

```
$ cat /etc/dhclient.conf
interface "eth0"
{
send dhcp-client-identifier 1:xx:xx:xx:xx:xx:xx;
send dhcp-lease-time 86400;
}
```

In the preceding file, the 1 in the **dhcp-client-identifier** specifies an Ethernet network and ***xx:xx:xx:xx:xx:xx*** is the *MAC address* (page 1092) of the device controlling that interface. See page 454 for instructions on how to display a MAC address. The **dhcp-lease-time** is the duration, in seconds, of the lease on the IP address. While the client is connected to the network, dhclient automatically renews the lease each time half of the lease is up. A lease time of 86,400 seconds (or one day) is a reasonable choice for a workstation.

DHCP SERVER

The DHCP server maintains a list of IP addresses and other configuration parameters. When requested to do so, the DHCP server provides configuration parameters to a client.

PREREQUISITES

Install the following package:

- **dhcp**

Run chkconfig to cause **dhcpd** to start when the system enters multiuser mode:

```
# /sbin/chkconfig dhcpd on
```

Start **dhcpd**:

```
# /sbin/service dhcpd start
```

dhcpd: THE DHCP DAEMON

A simple DHCP server allows you to add clients to a network without maintaining a list of assigned IP addresses. A simple network, such as a home LAN sharing an Internet connection, can use DHCP to assign a dynamic IP address to almost all nodes. The exceptions are servers and routers, which must be at known network locations to be able to receive connections. If servers and routers are configured without DHCP, you can specify a simple DHCP server configuration in **/etc/dhcp/dhcpd.conf** (*FEDORA*) or **/etc/dhcpd.conf** (*RHEL*):

```
$ cat /etc/dhcp/dhcpd.conf
default-lease-time 600;
max-lease-time 86400;

option subnet-mask 255.255.255.0;
option broadcast-address 192.168.1.255;
option routers 192.168.1.1;
option domain-name-servers 192.168.1.1;

subnet 192.168.1.0 netmask 255.255.255.0 {
   range 192.168.1.2 192.168.1.200;
}
```

The preceding configuration file specifies a LAN where the router and DNS are both located on **192.168.1.1**. The **default-lease-time** specifies the number of seconds the dynamic IP lease will remain valid if the client does not specify a duration. The **max-lease-time** is the maximum time allowed for a lease.

The information in the **option** lines is sent to each client when it connects. The names following the word **option** specify what the following argument represents. For example, the **option broadcast-address** line specifies the broadcast address of the network. The **routers** and **domain-name-servers** options can be followed by multiple values separated by commas.

The **subnet** section includes a **range** line that specifies the range of IP addresses that the DHCP server can assign. If you define multiple subnets, you can define options, such as **subnet-mask**, inside the **subnet** section. Options defined outside all **subnet** sections are global and apply to all subnets.

The preceding configuration file assigns addresses in the range between 192.168.1.2 and 192.168.1.200. The DHCP server starts at the bottom (*FEDORA*) or top (*RHEL*) of this range and attempts to assign a new IP address to each new client. Once the DHCP server reaches the top/bottom of the range, it starts reassigning IP addresses that have been used in the past, but are not currently in use. If you have fewer systems than IP addresses, the IP address of each system should remain fairly constant. You cannot use the same IP address for more than one system at a time.

Once you have configured a DHCP server, you can start (or restart) it by using the **dhcpd** init script:

```
# /sbin/service dhcpd restart
```

Once the server is running, clients configured to obtain an IP address from the server using DHCP should be able to do so.

STATIC IP ADDRESSES

As mentioned earlier, routers and servers typically require static IP addresses. While you can manually configure IP addresses for these systems, it may be more convenient to have the DHCP server provide them with static IP addresses.

When a system that requires a specific static IP address connects to the network and contacts the DHCP server, the server needs a way to identify the system so the server can assign the proper IP address to the system. The DHCP server uses the *MAC address* (page 1092) of the system's Ethernet card (NIC) as an identifier. When you set up the server, you must know the MAC address of each system that requires a static IP address.

Displaying a MAC address

You can use ifconfig to display the MAC addresses of the Ethernet cards (NICs) in a system. In the following example, the MAC addresses are the colon-separated series of hexadecimal number pairs following **HWaddr**:

```
$ /sbin/ifconfig | grep -i hwaddr
eth0      Link encap:Ethernet  HWaddr BA:DF:00:DF:C0:FF
eth1      Link encap:Ethernet  HWaddr 00:02:B3:41:35:98
```

Run ifconfig on each system that requires a static IP address. Once you have determined the MAC address of each of these systems, you can add a **host** section to the **/etc/dhcp/dhcpd.conf** file for each system, instructing the DHCP server to assign a specific address to the system. The following **host** section assigns the address **192.168.1.1** to the system with the MAC address of **BA:DF:00:DF:C0:FF**:

```
$ cat /etc/dhcp/dhcpd.conf
...
host router {
   hardware ethernet BA:DF:00:DF:C0:FF;
   fixed-address 192.168.1.1;
   option host-name router;
}
```

The name following **host** is used internally by **dhcpd**. The name specified after **option host-name** is passed to the client and can be a hostname or an FQDN.

After making changes to **dhcpd.conf**, restart **dhcpd** using the service utility and the **dhcpd** init script (page 453).

nsswitch.conf: WHICH SERVICE TO LOOK AT FIRST

With the advent of NIS and DNS, finding user and system information was no longer a simple matter of searching a local file. Where once you looked in **/etc/passwd** to get user information and in **/etc/hosts** to find system address information, you can now use several methods to find this type of information. The **/etc/nsswitch.conf** (name service switch configuration) file specifies the methods to use and the order in which to use them when looking for a certain type of information. You can also specify what action the system takes based on whether a method works or fails.

Format Each line in **nsswitch.conf** specifies how to search for a piece of information, such as a user's password. A line in **nsswitch.conf** has the following format:

info: ***method*** *[[action]] [**method** [[action]]...]*

where ***info*** is the type of information that the line describes, ***method*** is the method used to find the information, and ***action*** is the response to the return status of the preceding ***method***. The action is enclosed within square brackets.

How nsswitch.conf Works

When called upon to supply information that **nsswitch.conf** describes, the system examines the line with the appropriate ***info*** field. It uses the methods specified on the line starting with the method on the left. By default, when it finds the desired information, the system stops searching. Without an ***action*** specification, when a method fails to return a result, the system tries the next method. It is possible for the search to end without finding the requested information.

INFORMATION

The **nsswitch.conf** file commonly controls searches for users (in **passwd**), passwords (in **shadow**), host IP addresses, and group information. The following list describes most of the types of information (***info*** in the format discussed earlier) that **nsswitch.conf** controls searches for.

automount	Automount (**/etc/auto.master** and **/etc/auto.misc**; page 744)
bootparams	Diskless and other booting options (See the **bootparam** man page.)
ethers	MAC address (page 1092)
group	Groups of users (**/etc/group**; page 472)
hosts	System information (**/etc/hosts**; page 472)
netgroup	Netgroup information (**/etc/netgroup**; page 474)
networks	Network information (**/etc/networks**)
passwd	User information (**/etc/passwd**; page 475)
protocols	Protocol information (**/etc/protocols**; page 476)
publickey	Used for NFS running in secure mode
rpc	RPC names and numbers (**/etc/rpc**; page 477)
services	Services information (**/etc/services**; page 477)
shadow	Shadow password information (**/etc/shadow**; page 477)

METHODS

Following is a list of the types of information that **nsswitch.conf** controls searches for (***method*** in the syntax shown on page 455). For each type of information, you can specify one or more of the following methods:[1]

files	Searches local files such as **/etc/passwd** and **/etc/hosts**
nis	Searches the NIS database; **yp** is an alias for **nis**
dns	Queries the DNS (**hosts** queries only)
compat	**±** syntax in **passwd, group**, and **shadow** files (page 458)

SEARCH ORDER

The information provided by two or more methods may overlap: For example, **files** and **nis** may each provide password information for the same user. With overlapping information, you need to consider which method you want to be authoritative (take precedence) and then put that method at the left of the list of methods.

The default **nsswitch.conf** file lists methods without actions, assuming no overlap (which is normal). In this case, the order is not critical: When one method fails, the system goes to the next one; all that is lost is a little time. Order becomes critical when you use actions between methods or when overlapping entries differ.

1. There are other, less commonly used methods. See the default **/etc/nsswitch.conf** file and the **nsswitch.conf** man page for more information. Although NIS+ belongs in this list, it is not implemented for Linux and is not discussed in this book.

The first of the following lines from **nsswitch.conf** causes the system to search for password information in **/etc/passwd** and, if that fails, to use NIS to find the information. If the user you are looking for is listed in both places, the information in the local file would be used—it would be authoritative. The second line uses NIS; if that fails, it searches **/etc/hosts;** if that fails, it checks with DNS to find host information.

```
passwd          files nis
hosts           nis files dns
```

ACTION ITEMS

Each method can optionally be followed by an action item that specifies what to do if the method succeeds or fails for any of a number of reasons. An action item has the following format:

*[[!]STATUS=**action**]*

where the opening and closing square brackets are part of the format and do not indicate that the contents are optional; ***STATUS*** (by convention uppercase although it is not case sensitive) is the status being tested for; and ***action*** is the action to be taken if ***STATUS*** matches the status returned by the preceding method. The leading exclamation point (**!**) is optional and negates the status.

STATUS ***STATUS*** may have the following values:

NOTFOUND—The method worked but the value being searched for was not found. Default action is **continue**.

SUCCESS—The method worked and the value being searched for was found; no error was returned. Default action is **return**.

UNAVAIL—The method failed because it is permanently unavailable. For example, the required file may not be accessible or the required server may be down. Default action is **continue**.

TRYAGAIN—The method failed because it was temporarily unavailable. For example, a file may be locked or a server overloaded. Default action is **continue**.

action Values for ***action*** are as follows:

return—Returns to the calling routine with or without a value.

continue—Continues with the next method. Any returned value is overwritten by a value found by the next method.

Example The following line from **nsswitch.conf** causes the system first to use DNS to search for the IP address of a given host. The action item following the DNS method tests whether the status returned by the method is not (**!**) UNAVAIL.

```
hosts           dns [!UNAVAIL=return] files
```

The system takes the action associated with the ***STATUS*** (**return**) if the DNS method does not return UNAVAIL (**!**UNAVAIL)—that is, if DNS returns SUCCESS, NOTFOUND, or TRYAGAIN. As a consequence, the following method (**files**) is

used only when the DNS server is unavailable: If the DNS server is *not un*available (read the two negatives as "is available"), the search returns the domain name or reports that the domain name was not found. The search uses the **files** method (check the local **/etc/hosts** file) only if the server is not available.

COMPAT METHOD: ± IN passwd, group, AND shadow FILES

You can put special codes in the **/etc/passwd**, **/etc/group**, and **/etc/shadow** files that cause the system, when you specify the **compat** method in **nsswitch.conf**, to combine and modify entries in the local files and the NIS maps. That is, a plus sign (**+**) at the beginning of a line in one of these files adds NIS information; a minus sign (**–**) removes information.

For example, to use these codes in the **passwd** file, specify **passwd: compat** in **nsswitch.conf**. The system then goes through the **passwd** file in order, adding or removing the appropriate NIS entries when it reaches each line that starts with a **+** or **–**.

Although you can put a plus sign at the end of the **passwd** file, specify **passwd: compat** in **nsswitch.conf** to search the local **passwd** file, and then go through the NIS map, it is more efficient to put **passwd: file nis** in **nsswitch.conf** and not modify the **passwd** file.