Chapter 15

Applying Ownership and Permissions

✓ **Objective 2.5: Given a scenario, apply the appropriate access controls.**

Preventing unauthorized access to files and directories is a major part of any Linux administrator's job. Linux provides a few different methods for protecting files, which can make security a little bit complicated. This chapter dives into Linux file and directory security and demonstrates how to implement it on your Linux system. First, you'll see how Linux assigns ownership to files and directories. Then, the chapter discusses the basic file and directory security features that have been available since the dawn of Linux. Following that is an examination of a couple of newer methods for adding more protection to files and applications on your system. A brief overview describes how Linux handles administrator privileges when required to do work on the system.

Looking at File and Directory Permissions

The core security feature of Linux is file and directory permissions. Linux accomplishes that by assigning each file and directory an owner and allowing that owner to set the basic security settings to control access to the file or directory. The following sections walk through how Linux handles ownership of files and directories as well as the basic permission settings that you can assign to any file or directory on your Linux system.

Understanding Ownership

Linux uses a three-tiered approach to protecting files and directories:

Owner: In the Linux system, each file and directory is assigned to a single owner. The Linux system administrator can assign the owner specific privileges to the file or directory.

Group: The Linux system also assigns each file and directory to a single group of users. The administrator can then assign that group privileges that are specific to the file or directory and that differ from the owner privileges.

Others: This category of permissions is assigned to any user account that is not the owner or in the assigned user group.

You can view the assigned owner and group for a file or directory by adding the `-l` option to the `ls` command, as shown in Listing 15.1.

Listing 15.1: Viewing file owner and group settings

```
$ ls -l
total 12
-rw-rw-r--  1 Rich sales 1521 Jan 19 15:38 customers.txt
-rw-r--r--  1 Christine sales  479 Jan 19 15:37 research.txt
-rw-r--r--  1 Christine sales  696 Jan 19 15:37 salesdata.txt
$
```

In Listing 15.1, the first column, `-rw-rw-r--`, defines the access permissions assigned to the owner, group, and others. That will be discussed later in the chapter in the section "Controlling Access Permissions." The third column in Listing 15.1, the `Rich` or `Christine` value, shows the user account assigned as the owner of the file. The fourth column, `sales`, shows the group assigned to the file.

WARNING

Many Linux distributions (such as both Ubuntu and Rocky Linux) assign each user account to a separate group with the same name as the user account. This helps prevent accidental sharing of files. However, it can also make things a little confusing when you're working with owner and group permissions and you see the same name appear in both columns. Be careful when working in this type of environment.

When a user creates a file or directory, by default the Linux system automatically assigns that user as the owner and uses the primary group the user belongs to as the group for the file or directory. You can change the default owner and group assigned to files and directories by using Linux commands. The following sections show how to do that.

Changing File or Directory Ownership

The root user account can change the owner assigned to a file or directory by using the `chown` command. The `chown` command format looks like this:

```
chown [options] newowner filenames
```

The *`newowner`* parameter is the username of the new owner to assign to the file or directory, and *`filenames`* is the name of the file or directory to change. You can specify more than one file or directory by placing a space between each file or directory name:

```
$ sudo chown Christine customers.txt
$ ls -l
total 12
-rw-rw-r-- 1 Christine sales 1521 Jan 19 15:38 customers.txt
-rw-r--r-- 1 Christine sales  479 Jan 19 15:37 research.txt
-rw-r--r-- 1 Christine sales  696 Jan 19 15:37 salesdata.txt
$
```

There are a few command-line options available for the `chown` command, but they are mostly obscure and not used much. One that may be helpful for you is the `-R` option, which recursively changes the owner of all files under the specified directory.

Changing the File or Directory Group

The file or directory owner, or the root user account, can change the group assigned to the file or directory by using the `chgrp` command. The `chgrp` command uses this format:

```
chgrp [options] newgroup filenames
```

The *`newgroup`* parameter is the name of the new user group assigned to the file or directory, and the *`filenames`* parameter is the name of the file or directory to change. If you're the owner of the file, you can only change the group to a group that you belong to. The root user account can change the group to any group on the system:

```
$ sudo chgrp marketing customers.txt
$ ls -l
total 12
-rw-rw-r-- 1 Christine marketing 1521 Jan 19 15:38 customers.txt
-rw-r--r-- 1 Christine sales      479 Jan 19 15:37 research.txt
-rw-r--r-- 1 Christine sales      696 Jan 19 15:37 salesdata.txt
$
```

NOTE

The `chown` command allows you to change both the owner and the group assigned to a file or directory at the same time using this format:

```
chown newowner:newgroup filenames
```

This is often preferred over using the separate `chgrp` command.

Controlling Access Permissions

After you've established the file or directory owner and group, you can assign specific permissions to each. Linux uses three types of permission controls:

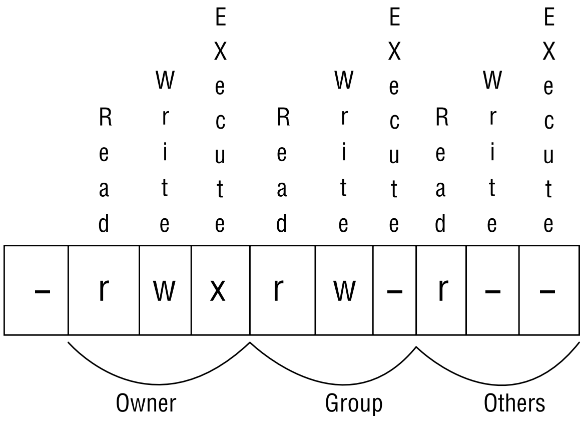
Read: The ability to access the data stored in the file or directory

Write: The ability to modify the data stored in the file or directory

Execute: The ability to run the file on the system, or the ability to list the files contained in the directory

You can assign each tier of protection (owner, group, and others) different read, write, and execute permissions. This creates a set of nine different permissions that are assigned to each file and directory on the Linux system. The nine permissions appear in the `ls` output as the first column of information when you use the `-l` option, as shown in Listing 15.1. Figure 15.1 shows the order in which the permissions are displayed in the `ls` output.

FIGURE 15.1 File and directory permissions as displayed in the `ls` output



In Figure 15.1, the first character denotes the object type. A dash indicates a file, and a `d` indicates a directory.

The next three characters denote the owner permissions in the order of read, write, and execute. A dash indicates the permission is not set, whereas the `r`, `w`, or `x` indicates the read, write, or execute permission is set. In the example in Listing 15.1, all three files use `rw-` for the owner permissions, which means the owner has permissions to read and write to the file but cannot execute, or run, the file. This is common with data files.

The second set of three characters denotes the group permissions for the file or directory. Again, this uses the read, write, and execute order, with a dash indicating the permission is not set. After making the change to the `customers.txt` file for the marketing group, the `sales` group can only read the `research.txt` and `salesdata.txt` files, but the `marketing` group can both read and write the `customers.txt` file.

Finally, the third set of three characters denotes the permissions assigned to user accounts that are not the owner or a member of the group assigned to the file or directory. The same order of read, write, and execute is used. In the Listing 15.1 examples, other user accounts on the system can read the files but not write or execute them.

Either the root user account or the owner of the file or directory can change the assigned permissions by using the `chmod` command.

The format of the `chmod` command can be somewhat confusing. It uses two different modes for denoting the read, write, and execute permission settings for the owner, group, and others. Both modes allow you to define the same sets of permissions, so there's no reason to use one mode over the other.

In *symbolic mode*, you denote permissions by using a letter code for the owner (`u`), group (`g`), others (`o`), or all (`a`) and another letter code for the read (`r`), write (`w`), or execute (`x`) permission. The two codes are separated with a plus sign (+) if you want to add the permission, a minus sign (-) to remove the permission, or an equal sign (=) to set the permission as the only permission. Listing 15.2 shows an example of this.

Listing 15.2: Changing file permissions

```
$ chmod g-w customers.txt
$ ls -al
total 12
-rw-r--r--  1 Christine marketing 1521 Jan 19 15:38 customers.txt
-rw-r--r--  1 Christine sales      479 Jan 19 15:37 research.txt
-rw-r--r--  1 Christine sales      696 Jan 19 15:37 salesdata.txt
$
```

In Listing 15.2, the `g-w` code in the `chmod` command indicates that we are removing the write permission for the group from the `customers.txt` file.

You can combine letter codes for both to make multiple changes in a single `chmod` command, as shown in Listing 15.3.

Listing 15.3: Combining permission changes

```
$ chmod ug=rwx salesdata.txt
$ ls -l
total 12
-rw-rw-r-- 1 Christine marketing 1521 Jan 19 15:38 customers.txt
-rw-r--r-- 1 Christine sales      479 Jan 19 15:37 research.txt
-rwxrwxr-- 1 Christine sales      696 Jan 19 15:37 salesdata.txt
$
```

The `ug` code assigns the change to both the owner and the group, whereas the `rwx` code assigns the read, write, and execute permissions. We use the equal sign to set those permissions.

The second mode available in `chmod` is called *octal mode*. With octal mode, the nine permission bits are represented as three octal numbers, one each for the owner, group, and other permissions. Table 15.1 shows how the octal number matches the three symbolic mode permissions.

TABLE 15.1 Octal mode permissions

Octal value	Permission	Meaning
0	---	No permissions
1	--x	Execute only
2	-w-	Write only

Octal value	Permission	Meaning
3	`-wx`	Write and execute
4	`r--`	Read only
5	`r-x`	Read and execute
6	`rw-`	Read and write
7	`rwx`	Read, write, and execute

You must specify the three octal values in the owner, group, and others in the correct order, as shown in Listing 15.4.

Listing 15.4: Using octal mode to assign permissions

```
$ chmod 664 research.txt
$ ls -l
total 12
-rw-r--r-- 1 Christine marketing 1521 Jan 19 15:38 customers.txt
-rw-rw-r-- 1 Christine sales      479 Jan 19 15:37 research.txt
-rwxrwxr-- 1 Christine sales      696 Jan 19 15:37 salesdata.txt
$
```

The `664` octal mode set the owner and group permissions to read and write (`6`) but the others permission to read only (`4`). You can see the results from the `ls` output. This is a handy way to set all the permissions for a file or directory in a single command.

Exploring Special Permissions

There are three special permission bits that Linux uses for controlling the advanced behavior of files and directories.

The *Set User ID (SUID)* bit is used with executable files. It tells the Linux kernel to run the program with the permissions of the file owner and not the user account actually running the file. This feature is most commonly used in server applications that must run as the root user account to have access to all files on the system, even if the user launching the process is a standard user.

The SUID bit is indicated by an `s` in place of the execute permission letter for the file owner: `rwsr-xr-x`. The execute permission is assumed for the system to run the file. If the SUID bit is set on a file that doesn't have execute permission for the owner, it's indicated by an uppercase `S`.

To set the SUID bit for a file, in symbolic mode add `s` to the owner permissions, or in octal mode include a `4` at the start of the octal mode setting:

```
# chmod u+s myapp
# chmod 4750 myapp
```

The *Set Group ID (SGID*, also called *GUID)* bit works differently in files and directories. For files, it tells Linux to run the program file with the file's group permissions. It's indicated by an `s` in the group execute position: `rwxrwsr--`.

For directories, the SGID bit helps us create an environment where multiple users can share files. When a directory has the SGID bit set, any files users create in the directory are assigned the group of the directory and not that of the user. That way, all users in that group can have the same permissions as all the files in the shared directory.

To set the SGID bit, in symbolic mode add `s` to the group permissions, or in octal mode include a `2` at the start of the octal mode setting:

```
# chmod g+s /sales
# chmod 2660 /sales
```

Finally, the *sticky bit* is used to protect a file from being deleted by those who don't own it, even if they belong to the group that has write permissions to the file. The sticky bit is denoted by a `t` in the execute bit position for others: `rwxrw-r-t`.

The sticky bit is often used on directories shared by groups. The group members have read and write access to the data files contained in the directory, but only the file owners can remove files from the shared directory.

To set the sticky bit, in symbolic mode add `t` to the owner permissions, or in octal mode include a `1` at the start of the octal mode setting:

```
# chmod o+t /sales
# chmod 1777 /sales
```

Managing Default Permissions

When a user creates a new file or directory, the Linux system assigns it a default owner, group, and permissions. The default owner, as expected, is the user who created the file. The default group is the owner's primary group.

The *user mask* feature defines the default permissions Linux assigns to the file or directory. The user mask is an octal value that represents the bits to be removed from the octal mode `666` permissions for files or the octal mode `777` permissions for directories.

The user mask value is set with the `umask` command. You can view your current `umask` setting by simply entering the command by itself on the command line:

```
$ umask
0022
$
```

The output of the `umask` command shows four octal values. The first octal value represents the mask for the SUID (`4`), GUID (`2`), and sticky (`1`) bits assigned to files and directories you create. The next three octal values mask the owner, group, and others permission settings.

The mask is a bitwise mask applied to the permission bits on the file or directory. Any bit that's set in the mask is removed from the permissions for the file or directory. If a bit isn't set, the mask doesn't change the setting. Table 15.2 demonstrates how the `umask` values work in practice when creating files and directories on your Linux system.

TABLE 15.2 Results from common `umask` values for files and directories

`umask`	**Created files**	**Created directories**
`000`	`666 (rw-rw-rw-)`	`777 (rwxrwxrwx)`
`002`	`664 (rw-rw-r--)`	`775 (rwxrwxr-x)`
`022`	`644 (rw-r--r--)`	`755 (rwxr-xr-x)`
`027`	`640 (rw-r-----)`	`750 (rwxr-x---)`
`077`	`600 (rw-------)`	`700 (rwx------)`
`277`	`400 (r--------)`	`500 (r-x------)`

You can test this by creating a new file and directory on your Linux system:

```
$ mkdir test1
$ touch test2
$ ls -l
total 4
drwxr-xr-x 2 rich rich 4096 Jan 19 17:08 test1
-rw-r--r-- 1 rich rich    0 Jan 19 17:08 test2
$
```

The `umask` value of 0022 created the default file permissions of `rw-r--r--` , or octal `644`, on the `test2` file, and `rwx-r-xr-x`, or octal `755`, on the `test1` directory, as expected (note that the directory entry starts with a `d` in the permissions list).

You can change the default `umask` setting for your user account by using the `umask` command from the command line:

```
$ umask 027
$ touch test3
$ ls -l test3
-rw-r----- rich rich 0 Jan 19 17:12 test3
$
```

The default permissions for the new file have changed to match the `umask` setting.

NOTE

The `umask` value is normally set in a script that the Linux system runs at login time, such as in the `/etc/profile` file. If you override the setting from the command line, that will only apply for the duration of your session. You can override the system default `umask` setting by adding it to the `.bash_profile` file in your `$HOME` directory.

Access Control Lists

The basic Linux method of permissions has one drawback in that it's somewhat limited. You can only assign permissions for a file or directory to a single group or user account. In a complex business environment with different groups of people needing different permissions to files and directories, that approach doesn't work.

Linux developers have devised a more advanced method of file and directory security called an *access control list (ACL)*. The ACL allows you to specify a list of multiple users or groups and the permissions that are assigned to them. Just like the basic security method, ACL permissions use the same read, write, and execute permission bits, but now they can be assigned to multiple users and groups.

To use the ACL feature in Linux, you use the `setfacl` and `getfacl` commands. The `getfacl` command allows you to view the ACLs assigned to a file or directory, as shown in Listing 15.5.

Listing 15.5: Viewing ACLs for a file

```
$ touch test
$ ls -l
total 0
-rw-r----- 1 rich rich 0 Jan 19 17:33 test
$ getfacl test
# file: test
# owner: rich
# group: rich
user::rw-
group::r--
other::---
$
```

If you've only assigned basic security permissions to the file, those still appear in the `getfacl` output, as shown in Listing 15.5.

To assign permissions for additional users or groups, you use the `setfacl` command:

```
setfacl [options] rule filenames
```

The `setfacl` command allows you to modify the permissions assigned to a file or directory by using the `-m` option or to remove specific permissions using the `-x` option. You define the *`rule`* with three formats:

```
u[ser]:uid:perms
g[roup]:gid:perms
o[ther]::perms
```

To assign permissions for additional user accounts, use the user format; for additional groups, use the group format; and for others, use the other format. For the `uid` or `gid` value, you can use either the numerical user identification number or group identification number or the names. Here's an example:

```
$ setfacl -m g:sales:rw test
$ ls -l
total 0
-rw-rw----+ 1 rich rich 0 Jan 19 17:33 test
$
```

This example adds read and write permissions for the `sales` group to the `test` file. Notice that there's no output from the `setfacl` command. When you list the file, only the standard owner, group, and others permissions are shown, but a plus sign (+) is added to the permissions list. This indicates that the file has additional ACLs applied to it. To view the additional ACLs, use the `getfacl` command again:

```
$ getfacl test
# file: test
# owner: rich
# group: rich
user::rw-
group::r--
group:sales:rw-
mask::rw-
other::---
$
```

The `getfacl` output now shows that there are permissions assigned to two groups. The default file group (`rich`) is assigned read permissions, but now the `sales` group has read and write permissions to the file. To remove the permissions, use the `-x` option:

```
$ setfacl -x g:sales test
$ getfacl test
# file: test
# owner: rich
# group: rich
```

```
user::rw-
group::r--
mask::r--
other::---

$
```

Linux also allows you to set a default ACL on a directory that is automatically inherited by any file created in the directory. This feature is called *inheritance*.

To create a default ACL on a directory, start the rule with a `d:` followed by the normal rule definition. That looks like this:

```
$ sudo setfacl -m d:g:sales:rw /shared/sales
```

This example assigns the read and write permissions to the `sales` group for the `/shared/sales` directory. Now all files created in that folder will automatically be assigned read and write permissions for the sales group.

Context-Based Permissions

Both the original permissions method and the advanced ACL method of assigning permissions to files and directories are called *discretionary access control (DAC)* methods. The permission is set at the discretion of the file or directory owner. There's nothing an administrator can do to prevent users from granting full permission to others on all the files in their directories.

To provide complete protection of your Linux system, it helps to utilize some type of *mandatory access control (MAC)* method. MAC methods allow the system administrator to define security based on the context of an object in the Linux system to override permissions set by file and directory owners. MAC methods provide rules for administrators to restrict access to files and directories not only to users but also to applications running on the system.

NOTE

You may also see the term *role-based access control (RBAC)* used in security literature. The RBAC method is a subcategory of MAC, basing security permissions on the roles users and processes play in the Linux system.

There are currently two popular MAC implementations in Linux:

- SELinux for Red Hat–based systems
- AppArmor for the Ubuntu system

The following sections provide more detail on using SELinux and AppArmor in your Linux environment.

Using SELinux

The *Security-Enhanced Linux (SELinux)* application is a project of the U.S. National Security Agency (NSA) and has been integrated into the Linux kernel since version 2.6.x. It is now a standard part of Red Hat–based Linux distributions, such as Fedora, Rocky, and CentOS, and an optional installation for Debian-based distributions.

SELinux implements MAC security by allowing you to set policy rules for controlling access between various types of objects on the Linux system, including users, files, directories, memory, network ports, and processes. Each time a user or process attempts to access an object on the Linux system, SELinux intercepts the attempt and evaluates it against the defined policy rules.

Enabling SELinux

The `/etc/selinux/config` file controls the basic operation of SELinux. There are two primary settings that you need to specify:

SELINUX: This setting determines the operation of SELinux. Set it to `enforcing` to enable the policy rules on the system and to block any unauthorized access. When you set it to `permissive`, SELinux monitors policy rules and logs any policy violations but doesn't enforce them. The `disabled` setting value completely disables SELinux from monitoring actions on the system.

SELINUXTYPE: This setting determines which policy rules are enforced. The `targeted` setting is the default and only enforces network daemon policy rules. The `minimum` setting only enforces policy rules on specified processes. The `mls` setting uses multilayer security, providing advanced policies following the Bell–LaPadula model of security control, which is mandated by most U.S. government and military environments that require high security. It uses security classifications such as top secret, unclassified, and public. The `strict` setting enforces policy rules for all daemons but is not recommended for use anymore.

To change the state of SELinux, you can also use the `setenforce` utility from the command line. However, you can only use the utility to change SELinux between enforcing and permissive modes. To disable SELinux, you must make the change in the SELinux configuration file. To see the current mode of SELinux, use the `getenforce` utility:

```
$ sudo getenforce
Enforcing
$
```

For a more detailed listing of the SELinux status, use the `sestatus` utility:

```
$ sudo sestatus
SELinux status:                 enabled
SELinuxfs mount:                /sys/fs/selinux
SELinux root directory:         /etc/selinux
```

```
Loaded policy name:             targeted
Current mode:                   enforcing
Mode from config file:          enforcing
Policy MLS status:              enabled
Policy deny_unknown status:     allowed
Memory protection checking      actual (secure)
Max kernel policy version:      33
$
```

After you've enabled SELinux, it starts enforcing access rules on the objects defined in a set of policies. The next sections explain how SELinux policies work.

Understanding Security Context

SELinux labels each object on the system with a *security context*. The security context defines what policies SELinux applies to the object. The security context format is as follows:

```
user:role:type:level
```

The *user* and *role* attributes are used only in the multilayer security mode and can get quite complex. Systems running in the default targeted security mode only use the *type* attribute to set the object security type and control access based on that. The *level* attribute sets the security sensitivity level and clearance level. It is optional under the targeted security mode and is mostly used in highly secure environments.

To view the security context assigned to objects, add the `-Z` option to common Linux commands such as `id`, `ls`, `ps`, and `netstat`. For example, to view your user security context, use the following command:

```
$ id -Z
unconfined_u:unconfined_r:unconfined_t:s0-s0:c0.c1023
$
```

The `unconfined_u` user security context means the user account is not assigned to a specific security policy; likewise, the `unconfined_r` for the role and the `unconfined_t` for the type. The level security context of `s0-s0:c0.c1023` means the security and clearance levels for the object are also not set.

To view the security context for a file, use this:

```
$ ls -Z test1.txt
unconfined_u:object_r:user_home_t:s0 test1.txt
$
```

Again, the user attribute is `unconfined_u`, but now the `type` attribute is set to `user_home_t`. You can use this attribute in a security policy to set the access for files in each user account's `$HOME` directory.

To examine the security context assigned to a process, use the following command:

```
$ ps -axZ | grep sshd
```

```
system_u:system_r:sshd_t:s0-s0:c0.c1023 1029 ? Ss 0:00
 /usr/sbin/sshd [...]
$
```

The process required for the sshd application is set to the system_u user security context and system_r role security context. These indicate that the process is system related. The type security context for the process is different, which means it can be controlled with separate policies.

NOTE You'll often see the security context referred to as a label in SELinux documentation and literature. SELinux must assign the label to each object on the system when it's first enabled, which can be a long process.

The semanage utility allows you to view and set the security context for user accounts on the system. For files and directories, the Linux system sets their security context when they are created, based on the security context of the parent directory. You can change the default security context assigned to a file by using the chcon or the restorecon utility.

The chcon format is as follows:

```
chcon -u newuser -r newrole -t newtype filename
```

The *newuser*, *newrole*, and *newtype* values define the new user, role, and type security contexts you want assigned to the specified file.

The restorecon utility restores the security context of a file or directory back to the default settings as defined in the policies. You can use the -R option to recursively restore the security context on all files under a specified directory.

WARNING The runcon utility allows you to start an application with a specified security context, but be careful. If an application starts without having access to any required configuration or logging files, strange things can, and usually will, happen.

Using Policies

SELinux controls access to system objects based on policies. In the targeted security mode, each policy defines what objects in a specific type security context can access objects in another type security context. This is called *type enforcement*.

For example, an application labeled with the type security context sshd_t is only allowed to access files labeled with the type security context sshd_t. This restricts access from the application to only certain files on the system.

SELinux maintains policies as text files in the /etc/selinux directory structure. For example, all policies for the targeted security mode are under the /etc/selinux/targeted directory.

Creating your own policies can be somewhat complicated. Fortunately, SELinux includes policy groups, called *modules*, that you can install as standard RPM packages. Use the semodule utility to list, install, and remove policy modules in your system.

To make things even easier, SELinux uses a method of enabling and disabling individual policies without having to modify a policy file. A *Boolean* is a switch that allows you to enable or disable a policy rule from the command line based on its policy name. To view the current setting of a policy, use the `getsebool` command:

```
$ getsebool antivirus_can_scan_system
antivirus_can_scan_system --> off
$
```

To view all of the policies for the system, include the `-a` option, as shown in Listing 15.6.

Listing 15.6: Using the `-a` option with the `getsebool` command

```
$ sudo getsebool -a
abrt_anon_write --> off
abrt_handle_event --> off
abrt_upload_watch_anon_write --> on
antivirus_can_scan_system --> off
antivirus_use_jit --> off
auditadm_exec_content --> on
authlogin_nsswitch_use_ldap --> off
authlogin_radius --> off
authlogin_yubikey --> off
awstats_purge_apache_log_files --> off
boinc_execmem --> on
cdrecord_read_content --> off
cluster_can_network_connect --> off
cluster_manage_all_files --> off
cluster_use_execmem --> off
cobbler_anon_write --> off
cobbler_can_network_connect --> off
cobbler_use_cifs --> off
cobbler_use_nfs --> off
collectd_tcp_network_connect --> off
condor_tcp_network_connect --> off
conman_can_network --> off
conman_use_nfs --> off
...
```

Listing 15.6 just shows a partial output from the `getsebool` command; there are lots of different policies installed by default in most Red Hat Linux environments.

To change the Boolean setting, use the `setsebool` command:

```
$ sudo setsebool antivirus_can_scan_system on
$ getsebool antivirus_can_scan_system
antivirus_can_scan_system --> on
$
```

This setting only applies to your current session. To make the change permanent, you must add the `-P` option to the command. Doing so gives you full control over the policy settings defined for SELinux.

TIP

Each time SELinux denies an event due to a security policy, it logs the action in the `/var/log/audit/audit.log` file. You can view this file to see what events security policies are blocking on your system. The `audit2allow` utility is a handy tool that can read an audit log entry and generate a policy rule that would allow the denied event. However, be careful with this tool, as there may have been a valid reason why the event was denied.

Using AppArmor

Debian-based Linux distributions commonly use the *AppArmor* MAC system. AppArmor isn't as complex or versatile as SELinux; it only controls the files and network ports that applications have access to.

WARNING

As of Ubuntu 18.04LTS, AppArmor is installed by default, but the utilities and profile packages aren't. Use `apt` to install the `apparmor-utils` and `apparmor-profiles` packages.

AppArmor also defines access based on policies but calls them *profiles*. Profiles are defined for each application in the `/etc/apparmor.d` directory structure. Normally, each application package installs its own profiles.

Each profile is a text file that defines the files and network ports the application is allowed to communicate with and the access permissions allowed for each. The name of the profile usually references the path to the application executable file, replacing the slashes with periods. For example, the profile name for the `mysqld` application program is called `usr.sbin.mysqld`.

NOTE

AppArmor profiles can use variables, called *tunables*, in the profile definition. The variables are then defined in files contained in the `/etc/apparmor.d/tunables` directory. This allows you to easily make changes to the variables to alter the behavior of a profile without having to modify the profile itself.

To determine the status of AppArmor on your Linux system, use the `aa-status` command, as shown in Listing 15.7.

Listing 15.7: The `aa-status` command output

```
$ sudo aa-status
apparmor module is loaded.
63 profiles are loaded.
42 profiles are in enforce mode.
   /snap/snapd/14066/usr/lib/snapd/snap-confine
   /snap/snapd/14066/usr/lib/snapd/snap-confine//mount-namespace-
capture-helper
   /snap/snapd/14295/usr/lib/snapd/snap-confine
   /snap/snapd/14295/usr/lib/snapd/snap-confine//mount-namespace-
capture-helper
   /usr/bin/evince
   /usr/bin/evince-previewer
   /usr/bin/evince-previewer//sanitized_helper
...
21 profiles are in complain mode.
   /usr/sbin/dnsmasq
   /usr/sbin/dnsmasq//libvirt_leaseshelper
   avahi-daemon
   chromium_browser
   chromium_browser//chromium_browser_sandbox
   chromium_browser//lsb:release
   chromium_browser//xdgsettings
   identd
...
3 processes have profiles defined.
3 processes are in enforce mode.
   /usr/sbin/cups-browsed (687)
   /usr/sbin/cupsd (679)
   /snap/snap-store/558/usr/bin/snap-store (1795) snap.snap-
store.ubuntu-software
0 processes are in complain mode.
0 processes are unconfined but have a profile defined.
$
```

The output from the `aa-status` command in Listing 15.7 shows all of the profiles in enforce, complain, or disabled status. After you've installed the `apparmor-utils` package, you have a few different useful commands for working with AppArmor at your disposal. You can view a listing of active network ports on your system that don't have a profile defined by using the `aa-unconfined` command:

```
$ sudo aa-unconfined
```

```
465 /lib/systemd/systemd-resolved not confined
747 /usr/sbin/avahi-daemon not confined
748 /usr/sbin/cupsd confined by '/usr/sbin/cupsd (enforce)'
804 /usr/sbin/cups-browsed confined by '/usr/sbin/cups-browsed (enforce)'
885 /usr/sbin/xrdp-sesman not confined
912 /sbin/dhclient confined by '/sbin/dhclient (enforce)'
935 /usr/sbin/xrdp not confined
982 /sbin/dhclient confined by '/sbin/dhclient (enforce)'
992 /usr/sbin/apache2 not confined
993 /usr/sbin/apache2 not confined
994 /usr/sbin/apache2 not confined
1094 /usr/sbin/mysqld confined by '/usr/sbin/mysqld (enforce)'
$
```

To turn off a specific profile, use the `aa-complain` command, which places the profile in complain mode:

```
$ sudo aa-complain /usr/sbin/tcpdump
Setting /usr/sbin/tcpdump to complain mode.
$
```

In complain mode, any violations of the profile will be logged but not blocked. If you want to completely disable an individual profile, use the `aa-disable` command:

```
$ sudo aa-disable /usr/sbin/tcpdump
Disabling /usr/sbin/tcpdump.
$
```

To turn a profile back on, use the `aa-enforce` command:

```
$ sudo aa-enforce /usr/sbin/tcpdump
Setting /usr/sbin/tcpdump to enforce mode.
$
```

Though not quite as versatile as SELinux, the AppArmor system provides a basic level of security protection against compromised applications on your Linux system.

Understanding Linux User Types

One of the more confusing topics in Linux is the issue of user types. In Linux, not all users are created equal, with some user accounts having different purposes, and therefore different permissions, than others. The following sections discuss the different types of Linux user accounts and how to change between them.

Types of User Accounts

While in Linux all user accounts are created the same way using the `useradd` utility (see Chapter 10, "Administering Users and Groups"), not all user accounts behave the same way. There are three basic types of user accounts in Linux:

Root: The root user account is the main administrator user account on the system. It is identified by being assigned the special user ID value of 0. The root user account has permission to access all files and directories on the system, regardless of any permission settings assigned.

Standard: Standard Linux user accounts are used to log into the system and perform standard tasks, such as running desktop applications or shell commands. Standard Linux users normally are assigned a `$HOME` directory, with permissions to store files and create subdirectories. Standard Linux users cannot access files outside their `$HOME` directory unless given permission by the file or directory owner. Most Linux distributions assign standard user account user IDs over 1000.

Service: Service Linux user accounts are used for applications that start in the background, such as network services like the Apache web server or MySQL database server. By setting the password value in the shadow file to an asterisk, these user accounts are restricted so that they cannot be used to log into the system. Also, the login shell defined in the `/etc/passwd` file is set to the `nologin` value to prevent access to a command shell. Service accounts normally have a user ID less than 1000.

NOTE Some Linux distributions, such as Ubuntu, don't allow you to log in directly as the root user account. Instead they rely on escalating privileges (discussed in the next section) to allow standard user accounts to perform administrative tasks.

Escalating Privileges

While the root user account has full access to the entire Linux system, it's generally considered a bad practice to log in as the root user account to perform system-related activities. There's no accountability for who logs in with the root user account, and providing the root user account password to multiple people in an organization can be dangerous.

Instead, most Linux administrators use privilege escalation to allow their standard Linux user account to run programs with the root administrator privileges. This is done using three different programs:

- *su*: The `su` command is short for *substitute user*. It allows a standard user account to run commands as another user account, including the root user account. To run the `su` command, the standard user must provide the password for the substitute user account. While this solves the problem of knowing who is performing the administrator task, it doesn't solve the problem of multiple people knowing the root user account password.

- *sudo*: The sudo command is short for *substitute user do*. It allows a standard user account to run any command as another user account, including the root user account. The sudo command prompts the user for their own password to validate who they are.
- *sudoedit*: The `sudoedit` command allows a standard user to open a file in a text editor with privileges of another user account, including the root user account. The `sudoedit` command also prompts the user for their own password to validate who they are.

Although the su command is somewhat self-explanatory, the sudo and `sudoedit` commands can be a bit confusing. Running a command with administrator privileges by supplying your own user password seems a bit odd.

Each Linux system uses a file that defines which users are allowed to run the sudo command, usually located at `/etc/sudoers`. The `sudoers` file contains a list of not only user accounts but also groups whose users are allowed administrator privileges. There are two common user groups that are used for these privileges. Debian-based distributions use the sudo group, and Red Hat–based distributions use the `wheel` group (short for *big wheel*).

WARNING

Never open the `sudoers` file using a standard editor. If multiple users open the `sudoers` file at the same time, odd things can happen and corrupt the file. The `visudo` command securely opens the file in an editor so that you can make changes.

Restricting Users and Files

Finally, there are two commands that you should know about that don't really have anything to do with file ownership or permissions but instead are related to user and file restrictions.

The `ulimit` command helps you restrict access to system resources for each user account. Listing 15.8 shows the output from running the `ulimit` command with the `-a` option, which displays the settings for the user account.

Listing 15.8: The `ulimit` command output

```
$ ulimit -a
core file size          (blocks, -c) 0
data seg size           (kbytes, -d) unlimited
scheduling priority             (-e) 0
file size               (blocks, -f) unlimited
pending signals                 (-i) 19567
max locked memory       (kbytes, -l) 16384
max memory size         (kbytes, -m) unlimited
open files                      (-n) 1024
pipe size            (512 bytes, -p) 8
```

```
POSIX message queues     (bytes, -q) 819200
real-time priority              (-r) 0
stack size              (kbytes, -s) 8192
cpu time               (seconds, -t) unlimited
max user processes              (-u) 19567
virtual memory          (kbytes, -v) unlimited
file locks                      (-x) unlimited
$
```

As a user account consumes system resources, it places a load on the system, but in CPU time and memory. If you're working in a multiuser Linux environment, you may need to place restrictions on how many resources each user account can consume. That's where the `ulimit` command comes in. Table 15.3 shows the command-line options you can use to restrict specific resources for the user account.

TABLE 15.3 The `ulimit` command options

Option	Description
`-a`	Lists the limits for the current user account
`-b`	Sets the maximum socket buffer size
`-c`	Sets the maximum core file size
`-d`	Sets the maximum data segment size for processes
`-e`	Sets the maximum allowed scheduling priority
`-f`	Sets the maximum file size allowed to be written
`-i`	Sets the maximum number of pending signals
`-k`	Sets the maximum number of kqueues that can be allocated
`-l`	Sets the maximum size of memory that can be locked
`-m`	Sets the maximum resident set size
`-n`	Sets the maximum number of open file descriptors
`-p`	Sets the maximum pipe size in 512k blocks
`-r`	Sets the maximum real-time scheduling priority value
`-s`	Sets the maximum stack size

Option	Description
`-t`	Sets the maximum amount of CPU time the user account is allowed
`-u`	Sets the maximum number of processes the user can run simultaneously
`-v`	Sets the maximum amount of virtual memory available to the user
`-x`	Sets the maximum number of file locks
`-P`	Sets the maximum number of pseudo-terminals the user account can log into
`-T`	Sets the maximum number of threads the user can have

As you can tell from Table 15.3, with the `ulimit` command the Linux administrator can place some pretty severe restrictions on just what an individual user account can do on the system.

You can also set restrictions on what users can or can't do with files and directories. File and directory *attributes* define actions that the filesystem can allow or block on the file or directory.

The `chattr` command modifies the attributes assigned to a file or directory. The format of the `chattr` command is:

```
chattr [ mode ] files...
```

The *mode* option defines what attributes to set or unset. Table 15.4 defines the different attributes available in Linux.

TABLE 15.4 Linux file and directory attributes

Attribute	Description
a	Can only open in append mode when writing.
A	The access time for the file is not modified when the file is open.
c	Automatically compress the file on the disk.
C	The file is not automatically copied on write for journaling filesystems.
d	Do not back up the file with the dump program.
D	All changes are synchronously written to disk without caching (applies to directories only).

TABLE 15.4 Linux file and directory attributes *(continued)*

Attribute	Description
`e`	Linux is using extents for mapping blocks on the disk.
`E`	The file is encrypted by the filesystem.
`F`	When applied to a directory, all path lookups in the directory are case-insensitive.
`i`	The file can't be modified or deleted.
`I`	Index the directory using a hash tree.
`j`	If the filesystem supports journaling, data written to the file is written to the journal before being written to the file.
`N`	The file contains data stored in the inode table itself.
`P`	When applied to a directory, files in the directory inherit the project ID of the directory.
`s`	If the file is deleted, its blocks are zeroed and written back to the disk.
`S`	Changes to the file are written synchronously to the disk and not stored in a buffer.
`t`	If the file contains a partial block fragment at the end, it will not be merged with other files.
`T`	The directory is deemed the top of a directory hierarchy for storage purposes.
`U`	Save the contents of the file when deleted, allowing for the undelete feature.
`V`	Apply file authentication to the file.

To assign an attribute to a file or directory you precede the attribute with a plus sign:

```
$ sudo chattr +i test1.txt
$ rm test1.txt
rm: cannot remove 'test1.txt': Operation not permitted
$
```

This example sets the immutable attribute (`i`) to a file, making it impossible to delete the file. This particular attribute can only be set by the root user account, thus the need to use the `sudo` command. After setting the attribute, the file can no longer be deleted.

To display the current attributes applied to a file or directory, use the `lsattr` command:

```
$ lsattr test1.txt
----i--------------- test1.txt
$
```

To remove the attribute, use the minus sign with the `chattr` command:

```
$ sudo chattr -i test1.txt
$ lsattr test1.txt
-------------------- test1.txt
$ rm test1.txt
$
```

WARNING

Not all filesystems support all the file and directory attributes. You'll need to consult the documentation for your specific filesystem to determine what file and directory attributes are available for you to use.

Exercise 15.1 walks through how to set up a simple shared directory where multiple user accounts can have read/write access to files.

Creating a Shared Directory

This exercise demonstrates how to use the GUID bit to create a directory where multiple users can both read and write to files.

1. Log into your Linux system and open a new command prompt.
2. Create the first test user account by using the command **`sudo useradd -m test1`**. Assign the test account a password by using the command **`sudo passwd test1`**.
3. Create a second test user account by using the command **`sudo useradd -m test2`**. Assign that test account a password by using the command **`sudo passwd test2`**.
4. Create a new group named sales by using the command **`sudo groupadd sales`**.
5. Add both test user accounts to the sales group by using the commands **`sudo usermod -G sales test1`** and **`sudo usermod -G sales test2`**. You can check your work by examining the group file using the command **`cat /etc/group | grep sales`**. You should see both the test1 and test2 user accounts listed as members of the group.
6. Create a new shared directory by using the command **`sudo mkdir /sales`**. Change the default group assigned to the directory by using the command **`sudo chgrp sales /sales`**. Grant members of the sales group write privileges to the `/sales` directory by using the command **`sudo chmod g+w /sales`**. Set the GUID bit for the `/sales` directory by using the command **`sudo chmod g+s /sales`**. This ensures that any files created in the `/sales` directory are assigned to the sales group.

7. Log out from the Linux system; then log in using the `test1` user account and open a new command prompt.
8. Change to the `/sales` directory using the command **`cd /sales`**.
9. Create a new text file using the command **`echo "This is a test" > testfile.txt`**. You can view the contents of the file using the command **`cat testfile.txt`**.
10. Log out from the Linux system; then log in using the `test2` user account and open a new command prompt.
11. Change to the `/sales` directory using the command **`cd /sales`**.
12. View the test file using the command **`cat testfile.txt`**.
13. Add to the test file using the command **`echo "This was added by the test2 user account" >> testfile.txt`**. View the contents of the file using the command **`cat testfile.txt`** to ensure that the `test2` user account also has write access to the file.
14. Log out from the Linux system; then log in using your normal user account.
15. Change to the `/sales` directory and see if you can view the contents of the test file by using the command **`cat testfile.txt`**.
16. Attempt to add text to the file by using the command **`echo "This was added by me" >> testfile.txt`**. This command should fail, as you're not a member of the `sales` group nor the owner of the file.

Summary

File and directory security is a major responsibility of all Linux administrators. The Linux system provides several layers of security that you can apply to files and directories to help keep them safe.

Linux assigns a set of read, write, and execute permissions to all files and directories on the system. You can define separate access settings for the file or directory owner, for a specific group defined for the Linux system, and for all other users on the system. The grouping of three access level settings and permissions provides for nine possible security settings applied to each file and directory. You can set those using the `chmod` command, using either symbolic mode or octal mode.

A more advanced method of file and directory security involves setting an access control list (ACL) for each file and directory. The ACL can define read, write, and execute permissions for multiple users or groups. The `setfacl` command allows you to set these permissions, and you use the `getfacl` command to view the current permissions.

The next level of security involves setting context-based permissions. Red Hat–based Linux distributions use the SELinux program to allow you to set policy rules that control access to files, directories, applications, and network ports based on the context of their use. For Debian-based Linux distributions, the AppArmor program provides advanced security for applications accessing files.

Linux handles security based on the user type. The root user account has full administrator privileges on the Linux system and can access any file, directory, or network port regardless of any security settings. Service user accounts are used to start and run applications that require access to a limited set of files and directories. Service user accounts usually can't log into the system from a terminal, nor can they open any type of command-line shell. The last type of user accounts is the standard user account. These accounts are for normal system users who need to log into a terminal and run applications.

Exam Essentials

Describe the basic level of file and directory security available in Linux. Linux provides basic file and directory security by utilizing three categories of read, write, and execute permissions. The file or directory owner is assigned one set of permissions, the primary group is assigned another set of permissions, and everyone else on the Linux system is assigned a third set of permissions. You can set the permissions in the three categories separately to control the amount of access the group members and others on the Linux system have.

Explain how to modify the permissions assigned to a file or directory. Linux uses the `chmod` command to assign permissions to files and directories. The `chmod` command uses two separate modes to assign permissions: symbolic mode and octal mode. Symbolic mode uses a single letter to identify the category for the owner (`u`), group (`g`), everyone else (`o`), and all (`a`). Following that, a plus sign, minus sign, or equal sign is used to indicate to add, remove, or set the permissions. The permissions are also indicated by a single letter for read (`r`), write (`w`), or execute (`x`) permissions. In octal mode an octal value is used to represent the three permissions for each category. The three octal values define the full set of permissions assigned to the file or directory.

Describe how Linux uses an access control list (ACL) to provide additional protection to files and directories. Linux allows you to set additional permissions for multiple users and groups to each file and directory. The `setfacl` command provides an interface for you to define read, write, and execute permissions for users or additional groups outside the owner and primary group assigned to the file or directory. The `getfacl` command allows you to view the additional permissions.

Describe how Linux uses context-based permissions for further file and directory security. Packages such as SELinux (for Red Hat–based distributions) and AppArmor (for Debian-based distributions) provide role-based mandatory access control (RBMAC) to enforce security permissions that override what the file or directory owner sets. The system administrator can define policies (or profiles in AppArmor) that are evaluated by the Linux kernel after any standard permissions or ACL rules are applied. You can fine-tune these permissions to control exactly what type of access the system allows to each individual file or directory.

Review Questions

1. What permissions can be applied to a file or directory? (Choose three.)
   - **A.** Read
   - **B.** Write
   - **C.** Delete
   - **D.** Modify
   - **E.** Execute
2. What user categories can be assigned permissions in Linux? (Choose three.)
   - **A.** Root
   - **B.** Owner
   - **C.** Group
   - **D.** Others
   - **E.** Department
3. Sam needs to allow standard users to run an application with root privileges. What special permissions bit should she apply to the application file?
   - **A.** The sticky bit
   - **B.** The SUID bit
   - **C.** The GUID bit
   - **D.** Execute
   - **E.** Write
4. What are the equivalent symbolic mode permissions for the octal mode value of 644?
   - **A.** `rwxrw-r--`
   - **B.** `-w--w--w-`
   - **C.** `-w-r--r--`
   - **D.** `rwxrw-rw-`
   - **E.** `rw-r–r--`
5. Fred was assigned the task of creating a new group on the company Linux server and now needs to assign permissions for that group to files and directories. What Linux utility should he use to change the group assigned to the files and directories? (Choose all that apply.)
   - **A.** `chgrp`
   - **B.** `chown`
   - **C.** `chmod`
   - **D.** `chage`
   - **E.** `ulimit`

6. Sally needs to view the ACL permissions assigned to a file on her Linux server. What command should she use?

A. `ls -Z`

B. `ls -l`

C. `getfacl`

D. `chmod`

E. `setfacl`

7. What SELinux mode tracks policy violations but doesn't enforce them?

A. Disabled

B. Enforcing

C. Targeted

D. Permissive

E. MLS

8. Ted is tasked with documenting the SELinux security context assigned to a group of files in a directory. What command should he use?

A. `getsebool`

B. `setsebool`

C. `ls -Z`

D. `getenforce`

E. `ls -l`

9. Mary is required to log into her Linux system as a standard user but needs to run an application with administrator privileges. What commands can she use to do that? (Choose all that apply.)

A. `su`

B. `wheel`

C. `visudo`

D. `sudo`

E. `adm`

10. What user groups are commonly used to assign privileges for group members to run applications as the administrator? (Choose two.)

A. `lp`

B. `adm`

C. `wheel`

D. `sudo`

E. `su`