Lab 16: Special Permissions, Links and File Locations

16.1 Introduction

This is Lab 16: Special Permissions. By performing this lab, students will learn about special permissions and different kinds of link files.

In this lab, you will perform the following tasks:

View files with special permissions

Create hard and soft links

16.2 Viewing Special Permissions

In this task, you will find and understand the purpose of special permissions beyond read, write, and execute.

16.2.1 Step 1

List the details of the `/tmp` and `/var/tmp` directories:

```
ls -ld /tmp
ls -ld /var/tmp
```

The output shows the permissions on these directories to be the same:

```
sysadmin@localhost:~$ ls -ld /tmp
drwxrwxrwt 2 root root 4096 Mar 14 17:34 /tmp
sysadmin@localhost:~$ ls -ld /var/tmp
drwxrwxrwt 2 root root 4096 Apr 19  2012 /var/tmp
sysadmin@localhost:~$
```

The `/tmp` and `/var/tmp` directories are read, write and executable for everyone. Besides the users' home directories, these two "temporary" directories are the locations in the filesystem where ordinary users can create new files or directories.

This does pose a problem: if all users can create new files, they are also able to delete existing files. This is because the write permission on a directory grants users the ability to add and delete files in a directory.

The `t` in the execute column for the *others* permissions indicates that this directory has the *sticky bit* permission set. This special permission means that even though everyone can add files in these directories, only the user who creates a file can delete that file.

The root user is not affected by this permission as that account can delete all files in the directory, regardless of ownership.

16.2.2 Step 2

View the permissions on the `/etc/shadow` file:

```
ls -l /etc/shadow
```

The output shows that the root user has read and write permissions, members of the shadow group have read permission, and others have no permission on this file:

```
sysadmin@localhost:~$ ls -l /etc/shadow
-rw-r----- 1 root shadow 838 Mar 14 17:34 /etc/shadow
sysadmin@localhost:~$
```

Like the other files found in the `/etc` directory, the `/etc/shadow` file contains host-specific configuration information. Specifically, the `/etc/shadow` file contains the encrypted passwords of all local user accounts and information about *password aging* (how long a password is good). Since this is very sensitive information, access to this file is limited to the root user and commands executing as root, as well as members of the `shadow` group.

When a user updates their password with the `passwd` command, the `passwd` command executes with a special permission called *setuid*. The setuid permission causes a file to execute as the user that owns the file, instead of the user that is actually running the command.

If you are coming from a Microsoft Windows background, you can think of setuid to be like the "Run as administrator" feature provided in Windows.

16.2.3 Step 3

View the permissions of the `/usr/bin/passwd` file:

```
ls -l /usr/bin/passwd
```

The listing of this file shows:

```
sysadmin@localhost:~$ ls -l /usr/bin/passwd
-rwsr-xr-x 5 root root 42824 Sep 12  2012 /usr/bin/passwd
sysadmin@localhost:~$
```

Notice the `s` in the user's execute permission column. This indicates that this file has the setuid permission set, so it executes as the user who owns it (`root`) instead of the user running the command.

Thus, the `passwd` command is able to update the `/etc/shadow` file, as it executes as the `root` user (recall that the root user can edit any file, regardless of the permissions on the file).

16.2.4 Step 4

View the permissions on the `/usr/bin/wall` command:

```
ls -l /usr/bin/wall
```

The output now shows the `s` in the execute column for the group:

```
sysadmin@localhost:~$ ls -l /usr/bin/wall
-rwxr-sr-x 5 root tty 18976 Jun 18  2014 /usr/bin/wall
sysadmin@localhost:~$
```

The `s` in the group execute column indicates that this file has the *setgid* permission set, so it executes as the group who owns it (`tty`) instead of the group of the user running the command. Thus, the `wall` command is able to write to all terminals (`ttys`) as it executes as the `tty` group.

Note: This is very similar to the setuid permission, but instead of running the command as the user owner of the program, the command runs as the group owner of the program.

So far, you've seen three types of special permissions: sticky bit on a directory, setuid on an executable file and setgid on an executable file. As you have seen, these three types exist on a typical system.

One more special permission type can be used; The setgid permission can also be applied to a directory.

If you see the `s` in the execute column for the group that owns directory, then the directory has the setgid permission. When a directory has the setgid permission, then any new file or directory created in that directory will automatically be owned by the group that owns the directory, not by the group of the user who created the file.

16.3 Hard and Soft Links

In this task, you will create and use hard and soft links.

16.3.1 Step 1

Change to your home directory:

```
cd
```

16.3.2 Step 2

Create a file named `source` containing the text "`data`" by using redirection:

```
echo "data" > source
```

16.3.3 Step 3

View the details and inode information of the `source` file:

```
ls -li source
```

The highlighted output shows that the file has an inode number of `2076`(this number will vary from one system to another) and a link count of `1`:

```
sysadmin@localhost:~$ ls -li source
2076 -rw-rw-r-- 1 sysadmin sysadmin 5 Apr 12 13:27 source
sysadmin@localhost:~$
```

The Linux operating system uses inodes to keep track of the information about a file. A directory entry associates an inode with a file name.

Creating a hard link creates another directory entry associated with an existing inode, and will increase the link count number.

Hard links allow you to have multiple names to refer to the same file. If any one of these names is removed, then the other names can still be used to refer to the file.

In fact, these other names can even be used to create additional links. All of these names are considered equivalent as they all refer to an existing inode.

Note: You cannot create hard links to directories. Also, a hard link to a file also must exist within the same filesystem (partition) as the file that it links to.

16.3.4 Step 4

Use the `ln` command to create a hard link. View the details and inode information of the source and new hard link file:

```
ln source hardlink
ls -li source hardlink
```

Notice that the `hardlink` file and the original `source` file share the same inode:

```
sysadmin@localhost:~$ ln source hardlink
sysadmin@localhost:~$ ls -li source hardlink
2076 -rw-rw-r-- 2 sysadmin sysadmin 5 Apr 12 13:27 hardlink
2076 -rw-rw-r-- 2 sysadmin sysadmin 5 Apr 12 13:27 source
sysadmin@localhost:~$
```

16.3.5 Step 5

Use the `ln` command to create another hard link to the `source` file. View the details and inode information of the source and new hard link files:

```
ln hardlink hardtoo
ls -li hardlink hardtoo source
```

Notice the output continues to show the hardlinks share the same inode and the link count increases on all links when a link is added:

```
sysadmin@localhost:~$ ln source hardtoo
sysadmin@localhost:~$ ls -li hardlink hardtoo source
2076 -rw-rw-r-- 3 sysadmin sysadmin 5 Apr 12 13:27 hardlink
2076 -rw-rw-r-- 3 sysadmin sysadmin 5 Apr 12 13:27 hardtoo
2076 -rw-rw-r-- 3 sysadmin sysadmin 5 Apr 12 13:27 source
sysadmin@localhost:~$
```

16.3.6 Step 6

Remove the last link that was created and list the `source` and `hardlink` files:

```
rm hardtoo
ls -li source hardlink
```

Notice that the link count decreases when one of the hard linked files is removed:

```
sysadmin@localhost:~$ rm hardtoo
sysadmin@localhost:~$ ls -li source hardlink
2076 -rw-rw-r-- 2 sysadmin sysadmin 5 Apr 12 13:27 hardlink
```

```
2076 -rw-rw-r-- 2 sysadmin sysadmin 5 Apr 12 13:27 source
sysadmin@localhost:~$
```

16.3.7 Step 7

Remove the `hardlink` file and list the `source` file details:

```
rm hardlink
ls -li source
```

```
sysadmin@localhost:~$ rm hardlink
sysadmin@localhost:~$ ls -li source
2076 -rw-rw-r-- 1 sysadmin sysadmin 5 Apr 12 13:27 source
sysadmin@localhost:~$
```

Another type of link that can be created is known as a symbolic link or soft link. Symbolic links do not increase the link count of files with which they are linked.

Symbolic link files have their own inode and type of file. Instead of linking and sharing an inode, they link to the file name. Unlike hard links, soft links can be linked to directories and can cross devices and partitions to their targets.

16.3.8 Step 8

Create a symbolic link to the `source` file and view the details of both files:

```
ln -s source softlink
ls -li source softlink
```

The highlighted output shows that the `softlink` and the `source` file have different inodes. Notice that the link type of file is created when making a symbolic link. The permissions of the link are irrelevant, as it is the permissions of the target file that determine access:

```
sysadmin@localhost:~$ ln -s source softlink
sysadmin@localhost:~$ ls -li source softlink
2086 lrwxrwxrwx 1 sysadmin sysadmin 6 Apr 12 13:56 softlink -> source
2076 -rw-rw-r-- 1 sysadmin sysadmin 5 Apr 12 13:27 source
sysadmin@localhost:~$
```

16.3.9 Step 9

Create a symbolic link to the `/proc` directory and display the link:

```
ln -s /proc crossdir
ls -l crossdir
```

The success of these commands shows that soft links can refer to directories, and they can cross from one filesystem to another, which are two things that hard links cannot do:

```
sysadmin@localhost:~$ ln -s /proc crossdir
```

```
sysadmin@localhost:~$ ls -l crossdir
lrwxrwxrwx 1 sysadmin sysadmin 5 Apr 12 14:00 crossdir -> /proc
sysadmin@localhost:~$
```