Linux Privilege Escalation

Linux File System Permissions

Leonardo Tamiano

Contents

1 MAN pages

The command `man` (short for "manual page") is used for accessing local documentation of common linux commands.

```
man ls
man ps
man df
man du
```

You can either download it locally or you can browser the online version at the following URL

- https://man7.org/linux/man-pages/

2 File System Permissions

Suppose we list out the files in a given directory with `ls`

```
$ ls -lha

drwxr-xr-x  3 leo  users 100 28 mag 20.03 .
drwxrwxrwt 18 root root  440 28 mag 20.03 ..
drwxr-xr-x  2 leo  users  40 28 mag 20.02 dir
-rwxr-xr-x  1 leo  users   0 28 mag 20.03 exec.sh
-rw-r--r--  1 leo  users   6 28 mag 20.02 test.txt
```

For each file we have the following information regarding permissions

- File permissions (`-rwxr-xr-x`)
- User that owns the file (`leo`)
- Group that owns the file (`users`)

Within each string of the form

```
drwxrwxrwt
drwxr-xr-x
-rw-r--r--
```

we find 10 letters, which can be either set or not set:

- The first specifies the `type of the file`.
  - Regular files -> "-"
  - Directories -> "d"
  - Links -> "l"
- Then we find three groups of three letters each.

  These specify the permissions for three types of users:
  - The first group specifies the permissions for the user who is also the owner of the file.
  - The second group specifies the permissions for the user who belongs to the group that owns the file.
  - The third group specifies the permissions for all other users. That is, for users who do not belong in the group that owns the file and are not the owner of the file themselves.

- Each group works in the same way.

  We have three distinct types of permissions:

  - Read permission (r)
  - Write permission (w)
  - Execute permission (x)

  There are minor variants in this. For example, when setting the SUID bit on an executable, the execute permission will be written as `s` instead of `x`.

```
1  -rwsr-sr-x  1 leo  users   0 28 mag 20.03 exec.sh
```

3 Reading File Permissions

File permissions can be read with the `ls` command.

Consider the following output

```
$ ls -lha

drwxr-xr-x  3 leo  users 100 28 mag 20.03 .
drwxrwxrwx 18 root root  440 28 mag 20.03 ..
drwxr-xr-x  2 leo  users  40 28 mag 20.02 dir
-rwxr-xr-x  1 leo  users   0 28 mag 20.03 exec.sh
-rw-r--r--  1 leo  users   6 28 mag 20.02 test.txt
```

We can analyze the output line by line:

- The first line is referring to the current directory (.).

  It is saying that:

  - It is a directory (d)
  - The user `leo` can read (r), write (w) and execute (x) on the directory.
  - Users beloging to the group `users` can read (r) and execute (x) but not write.
  - Any other user can read (r) and execute (x) but not write.

- The second line is referring to the previous directory (..).

  It is saying that:

  - It is a directory (d)
  - The user `root` can read (r), write (w) and execute (x) on the directory.
  - Users beloging to the group `root` can read (r), write (w) and execute (x) on the directory.
  - Any other users can read (r), write (x) and execute (x) on the directory.

- …

- The last line is referring to the resource test.txt.

  It is saying that:

  - It is a regular file (-)
  - The user `leo` can read (r), write(r), but not execute (x) the file.

- Users belonging to the group `users` can only read (r) the file.
- Any other users can also read (r) the file.

Special Cases

Sometimes it can happen to see for the last position the value of `t` instead of the value of `x`. This is called the "sticky bit". It implies that users in the group `root` who technically have the permissions to delete a file (write), cannot do it as long as the sticky bit is set.

Other times instead we see the value of `s` instead of the value of `x`. This instead has to do with the `SUID` and `GUID` permissions, which will be discussed later.

4 Updating File Permissions

The command that can be used for updating the permission on a given file is the chmod command.

Let's assume to start with the following permission

```
$ ls -lh
-rw-r--r-- 1 leo users   0 28 mag 20.26 exec.sh
```

It can be used as follows:

- Add execute (x) for all

```
$ chmod +x exec.sh
$ ls -lh
-rwxr-xr-x 1 leo users   0 28 mag 20.27 exec.sh
```

- Remove read (r) and execute (x) for other users

```
$ chmod o-xr exec.sh
$ ls -lh
-rwxr-x--- 1 leo users   0 28 mag 20.27 exec.sh
```

- Add write (w) for group that owns the file

```
$ chmod g+w exec.sh
$ ls -lh
-rwxrwx--- 1 leo users   0 28 mag 20.27 exec.sh
```

- If we want to make a change in an entire subfolder, we have to use the recursive (R) option

```
$ chmod -R o-rwx directory
```

To change the owner of the file we can use the chown command.

```
$ chown root test.sh
```

If you want to change the group you can use the syntax with the `:`.

```
$ chown <USER>:<GROUP> test.sh
$ chown root:root test.sh
$ ls -lh
$ -rwxrwxr-- 1 root root    0  2 giu 12.34 test.sh
```

Instead of using the identifiers `ugoa` with `rwx`, it is also possible to set permissions using a sequence of three numbers. This is because read (r), write (w) and execute (x) permissions can be represented using three bits.

Given that with three bits we can also represent the numbers from `0` to `7`, this means that each number can represent a specific set of permissions.

Specifically, we have the following association between bits, permissions and numbers.

```
0  ->  000  ->  ---
1  ->  001  ->  --x
2  ->  010  ->  -w-
3  ->  011  ->  -wx
4  ->  100  ->  r--
5  ->  101  ->  r-x
6  ->  110  ->  rw-
7  ->  111  ->  rwx
```

This means for example that 5 represents the read (r) and execute (x) permission bits, while 7 represent the entire read (r), write (w) and execute (x) bits.

Remember that we have three groups of users. This means that to fully specify the basic permissions for a file we will need three numbers, each going from `0` to `7`.

- Set rwx for owner, rw for group and r for others

```
$ chmod 764 exec.sh
$ ls -lh
-rwxrw-r-- 1 leo users   0 28 mag 20.27 exec.sh
```

- Set rw for owner, r for group and nothing for others

```
$ chmod 640 exec.sh
```

5 SUID and GUID

Executable files can have special permissions called `SUID` and `GUID`.

`SUID` stands for `Set User ID`.

Binaries that have this bit allow the binary to change user permission during its execution.

Specifically, it allows the program to be executed by anyone. Then, during its execution, it will change its permission and take the permissions of the owner of the file.

For example, consider the `passwd` program

```
$ ls -lh /usr/bin/passwd
-rwsr-xr-x 1 root root 80K  1 apr 12.19 /usr/bin/passwd
```

The executable flag is not (x) but rather (s). This is because the program has the SUID bit set. This means that durings its execution, at specific times, the program will change its roles in order to become root.

This change of permissions is done by using the function `setuid` offered by the standard C library. It is necessary since the `shadow` file is owned by `root` and only `root` can edit it.

github.com/shadow-maint/passwd.c

```
if (setuid (0) != 0) {
  (void) fputs (_("Cannot change ID to root.\n"), stderr);
  SYSLOG ((LOG_ERR, "can't setuid(0)"));
  closelog ();
  exit (E_NOPERM);
 }
if (spw_file_present ()) {
  update_shadow ();
 } else {
  update_noshadow ();
 }
```

```
$ ls -lh /etc/shadow
-rw------- 1 root root 1,3K 26 mag 17.19 /etc/shadow
```

To set a SUID bit we can use `chmod`

```
$ chmod u+s exec.sh
$ ls -lh
-rwsr-x--x 1 leo users   0 28 mag 20.27 exec.sh
```

The same idea of SUID also applies to GUID.

This time however instead of using the file owner permission, the program during its execution is allowed to obtain the permission of the group that owns the file.

To set a GUID bit we can use chmod

```
$ chmod g+s exec.sh
$ ls -lh
-rwxr-s--x 1 leo users 0 28 mag 20.27 exec.sh
```

6 Sudo

Finally, an important subystem that relates to permissions is the `sudo` functionality.

```
sudo -> SuperUserDO
```

The sudo utility allows to change user for the execution of specific commands. Once it is installed, we can check the way we can use it with `sudo -l`

```
Matching Defaults entries for homer on canape:
env_reset, mail_badpass,
secure_path=/usr/local/sbin\:/usr/local/bin\:/usr/sbin\:/usr/bin\:/sbin\:/bin\:/snap/bin

User homer may run the following commands on canape:
(root) /usr/bin/pip install *
```

```
env_reset, mail_badpass, secure_path=/usr/local/sbin\:/usr/local/bin\:/usr/sbin\:/usr/bin
    \:/sbin\:/bin

User richard may run the following commands on stratosphere:
(ALL) NOPASSWD: /usr/bin/python* /home/richard/test.py
```

```
Matching Defaults entries for genevieve on dab:
env_reset, mail_badpass,
secure_path=/usr/local/sbin\:/usr/local/bin\:/usr/sbin\:/usr/bin\:/sbin\:/bin\:/snap/bin

User genevieve may run the following commands on dab:
(root) /usr/bin/try_harder
```

```
User dorthi may run the following commands on Oz:
(ALL) NOPASSWD: /usr/bin/docker network inspect *
(ALL) NOPASSWD: /usr/bin/docker network ls
```

```
Matching Defaults entries for www-data on bashed:
env_reset, mail_badpass,
secure_path=/usr/local/sbin\:/usr/local/bin\:/usr/sbin\:/usr/bin\:/sbin\:/bin\:/snap/bin

User www-data may run the following commands on bashed:
(scriptmanager : scriptmanager) NOPASSWD: ALL
```

To use sudo we can do

```
sudo -u scriptmanager python3 -c 'import pty; pty.spawn("/bin/bash")'
```

The configuration file for sudo is found in `/etc/sudoers`.