Sticky, SUID, SGID – Special Privileges for Files and Directories

Talking Privileges

Read, write, and execute file privileges are explained in nearly every Unix or Linux manual. There is more, the notorious SUID bit, for example.

BY PATRICIA JUNG

One of the first lessons that a Linux newbie learns is the fact that they can only access the files and directories of a multiuser system if they have appropriate privileges. The owner (shortened to *u* for "*u*ser") of a file has certain privileges, the members of the owner group ("group") have certain privileges, as do all the other users of the system ("*o*thers"). You can type ***ls*** *-al file* in the command line to see what you can do with the file:

```
pjung@linux:~> ls -al /bin/ls
-rwxr-xr-x 1 root wheel 48832 ➥
Feb 17  2000 /bin/ls
```

In our example, the owner of the file */bin/ls* (that is *root*, as is typically the case for system tools such as *ls*) can ("*r*ead"), "*w*rite" (that is modify), and ("*e*xecute") the file. This first group of privileges, *rwx*, is followed by the details for the members of the *wheel* group. They are not allowed to modify */bin/ls* (instead of a *w* the middle privilege in the group has been replaced by a -), but they can read and execute the file.

The same applies to all other users of the system, and in the case of the last *x* bit, it is probably just as well – if the *x* were replaced by a -, normal users would not even be able to display directory contents, as they would not be able to run the *ls* command.

If you look at directory privileges:

```
pjung@linux:~> ls -ald /home
drwxr-xr-x 55 root root 2048 ➥
Sep  2 13:08 /home
```

you will probably not notice any big surprises, although the meaning of *rwx* has a slightly different emphasis.

r means you are allowed to look inside a directory, *w* means you can create files in the directory, and *x* means you can change to the directory. The *x* privilege is also required to open and execute files inside the directory along with being able to perform file operations with them.

Sticky Directories

If you look at the */tmp* directory:

```
pjung@linux:~> ls -ald /tmp
drwxrwxrwt 11 root root 6144 ➥
Oct 18 13:50 /tmp
```

You will note that the lessons in the beginners' manual do not cover all the details you might need. Suddenly, instead of the execute privilege for others, there is that small *t*.

This special privilege is usually referred to as the "sticky bit", and it really does make the directories sticky. The files stored in a sticky directory can only be deleted by their owners, even if other users have write privileges set for that directory.

Incidentally the right to change to the directory has not disappeared: *ls* simply displays a small *t* instead of the third *x*, if this privilege has been set. If the *x* privilege is not available, a capital *T* is displayed instead.

Non-privileged users can neither store nor delete files in directories they are not permitted to change to. If the privileges read *rwxrwxrwT*, only the user and group for this directory will be affected by the sticky bit.

What's the big deal about the sticky bit? Directories like */tmp* that are used for storing temporary files, should be available for write access to all the users on the system.

However, if everyone is allowed to create files in the directory, and the privileges *rwxrwxrwx* apply, everyone will be able to delete any files in the directory (Box 1) and that is definitely not what the administrator envisaged.

Group Therapy

Special privileges are not restricted to the third digit in the privilege triplet "others", they can also be applied to the owner and group triplets. If a so-called *s* bit is also applied to the group privileges, it affects any files written to the directory. No matter who creates the files, they are always assigned to the group that owns the directory. If group members are not permitted to make changes to a directory, *ls* reveals a capital *S* instead of a small letter.

Again this rather abstract concept can be explained by a practical example: If the system administrator assigns a number of users to a group, she will normally want to assign common file access to these members. However, most users will also be members of other groups – even if these are personal groups that only have one member, or simply the *users* group:

```
pjung@linux:~> groups
users uucp dialout audio video
```

The first group that the *groups* command outputs is the so-called primary group. The system adminstrator specifies the primary group when creating a user account; the group ID (in this example the group ID for *users* is *100*) is defined in the entry for the user account in */etc/passwd*:

```
pjung@linux:~> grep pjung /etc/passwd
pjung:x:500:100:Patricia Jung:/home/pjung:/bin/bash
```

The */etc/group* file keeps track of any group memberships:

```
pjung@linux:~> grep pjung /etc/group
uucp:x:14:uucp,fax,root,fnet,pjung
dialout:x:16:root,pjung
audio:x:17:root,pjung
video:x:33:pjung
```

If a user now creates some files, the system will take care of assigning them to the user's primary group. If the user wants to assign these files to a different group, she must use the *chgrp* command to do so. This is somewhat laborious, and not really practical within a working group, so the administrator will tend to assign the SGID ("*S*et *G*roup *ID*") bit to the parent directory to solve the problem (Box 2).

Under a Foreign Flag

The SUID bit, that is the *s* bit that is set in the owner privileges triplet, is an exception here as it does not apply to the directories, but exclusively to executable files. It ensures that Linux will not run the program with the rights of the user that calls it, but with the rights of the file's owner.

Thus, the *passwd* program, which is used for changing passwords, effectively

Box 1: The difference between *rwx* and *rwt*

In the following example, a non-privileged user assumes *root* privileges using the *su* command. Using *root* privileges she creates a directory called *foo* with the *mkdir* command and allows *a*ll users read, write, and execute privileges for the new directory. Now *root* uses the *touch* command to create a new (empty) file called *Hello* which only *root* can write to as the *ls -al* command shows.

```
pjung@linux:~> su
Password: root password
# mkdir foo
# chmod a+rwx foo
# ls -ald foo
drwxrwxrwx    2 root     root         4096 Oct 18 14:09 foo
# touch foo/Hello
# ls -al foo/Hello
-rw-r--r--    1 root     root            0 Oct 18 14:09 foo/Hello
# exit
```

As the non-privileged user has write access to the *foo* directory, she can delete *Hello* (following a prompt), although *root* did not actually assign write privileges for the file to her:

```
pjung@linux:~> rm foo/Hello
rm: Remove write-protected file "foo/Hello"? y
pjung@linux:~> ls -al foo/Hello
ls: foo/Hello: File or directory not found
```

Now, if *root* had applied a sticky bit to the *foo* directory...

```
pjung@linux:~> su
Password: root password
# chmod a+rwx,o+t foo
# ls -ald foo
drwxrwxrwt    2 root     root         4096 Oct 18 14:09 foo
# touch foo/Hello
# ls -al foo/Hello
-rw-r--r--    1 root     root            0 Oct 18 14:22 foo/Hello
# exit
```

... this would never have happened:

```
pjung@linux:~> rm foo/Hello
rm: Remove write-protected file "foo/Hello" ? y
rm: Remove (unlink) "foo/Hello" not possible: Operation not permitted
```

GLOSSARY

ls: *You can simply type ls ("list" to run the command /bin/ls, because the /bin directory is defined in your search path. If you additionally specify the -a flag, ls will also list hidden files. The option -l forces a "long list", which additionally includes the privileges. -d tells ls that you are not interested in the content of the directory passed to the command, but in the directory itself.*

grep: *This command line program searches for string (or to be more precise, a regular expression) in a file (or in the standard input data). If you do not change its default behavior by supplying additional options, the program outputs any lines containing the search string.*

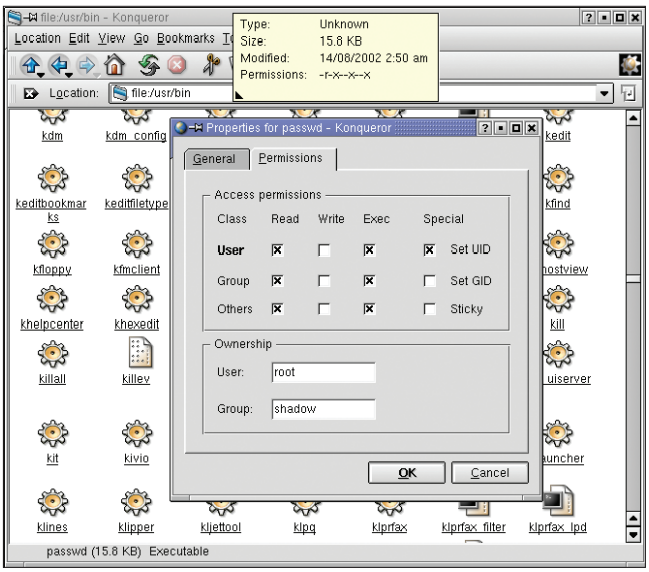

Figure 1: Konqueror does not mention the SUID bit in its bubble help – you need to access the "Properties" dialog box via the drop-down menu

uses *root* privileges, even if it is called by a non-privileged user:

```
pjung@linux:~>
ls -al /usr/bin/passwd
-r-sr-xr-x 1 root bin 8735
Feb 17  2000  /usr/bin/passwd
```

The reason: The password file needs to be edited to change a password. However, this file is not globally available for write access to prevent all kinds of mischief with other users passwords. If the *passwd* program were to be run with the callers privileges, as is normally the case, it would only be able to modify files for which the user had write access, and that definitely excludes the password file.

The SUID bit can help solve this problem by assigning *root* privileges to the *passwd* program while it is running, thus allowing it to change passwords.

The SUID bit is as dangerous as it is useful. If a program running with SUID privileges has not been programmed carefully, it may offer malevolent hackers an opportunity to attach a system.

It might even allow a process running with SUID privileges to execute additional programs with the same privileges, or even allow a user access to files that should not be visible to them. For security reasons it is a good idea to be careful when using the SUID bit. In the case of shell scripts, you should not use it at all.

The same principle that applies to user privileges also applies to the group privileges. If you set the SGID bit for executables, the processes will be run with the privileges of the owner group, the only condition being that the user must have execute access privileges for the file in question.

Setting Special Privileges

Whether you are permitting or removing *rwx* or *s* and *t* – the *chmod* ("*ch*ange *mod*e") is the tool to look for. Arithmetic operations +, - and = and apply as they do to read, write and executable privileges: *chmod u-s executable* removes the SUID privilege from a program, *chmod g+xs directory* assigns the SGID bit and the right to change directory to the owner groups, and *chmod o=rwxt directory* sets the privileges for others to *rwt*.

If the underlying *x* bit has not been assigned, *ls* will indicate this by displaying a capital letter *S* or *T*.

Of course special privileges can be assigned using octal notation. Just as the read privilege, *r*, is represented by a value of 4, the write privilege, *w*, by a value of 2, and the *x* bit by a 1, the numeric equivalents are as follows:

- 1 for the *t* bit (sticky bit),
- 2 for the SGID bit and
- 4 for the SUID bit.

To prevent conflicts with *rwx*, they are not added to the sum of the user, group or other privileges, but form a fourth and separate value: thus *chmod 1777 directory* will assign *rwxrwxrwt* to *directory*, just like *chmod u,g=rwx,o=rwxt*. The first number represents the sum of the special privileges, that is 1 for *t* in this case, the second digit represents the sum of the "normal" user privileges ($r+w+x=4+2+1=7$), the third the group privileges, and the last number represents other privileges.

To apply the SGID bit (*rwxrwsrwt*), you can issue the *chmod 3777 command*.

The privileges described earlier as applying to */usr/bin/passwd* (*r-sr-xr-x*) were most likely assigned using *chmod 4555 /usr/bin/passwd*. ■

Box 2: Directories with and without SGID bits

The user *pjung* can be a member of multiple groups. If she creates a directory (called *music*) in our example, it will automatically be assigned to her primary group *users*.

```
pjung@linux:~> mkdir music
pjung@linux:~> ls -ald music
drwxr-xr-x 2 pjung users  4096 Oct 18 15:23 music
```

Even she now assigns *music* to the *audio* group instead, the system will still assign the files created there, such as *wish.txt* to the primary group:

```
pjung@linux:~> chgrp audio music
pjung@linux:~> ls -ald music
drwxr-xr-x 2 pjung audio  4096 Oct 18 15:23 music
pjung@linux:~> touch music/wish.txt
pjung@linux:~> ls -al music/wish.txt
-rw-r--r-- 1 pjung users     0 Oct 18 15:26 wish.txt
```

The SGID bit needs to be for the directory *music* to ensure that any files created here will automatically be assigned to the *audio* group. This does not affect the ownership of any existing files:

```
pjung@linux:~> chmod g+s music
pjung@linux:~> ls -ald music
drwxr-sr-x 2 pjung audio  4096 Oct 18 15:26 music
pjung@linux:~> touch music/content.txt
pjung@linux:~> ls -l music
insgesamt 0
-rw-r--r-- 1 pjung audio     0 Oct 18 15:29 content.txt
-rw-r--r-- 1 pjung users     0 Oct 18 15:26 wish.txt
```