SUID, SGID Explained

Author: Unknown

We often neglect the basics of Linux such as the Suid, sgid and sticky bits. But they are really useful if we know to use them properly.

Below I am explaining it briefly with appropriate examples.

1) Sticky bit was used on executables in linux (which was used more often) so that they would remain in the memory more time after the initial execution, hoping they would be needed in the near future. But since today we have more sophisticated memory accessing techniques and the bottleneck related to primary memory is diminishing, the sticky bit is not used today for this. Instead, it is used on folders, to imply that a file or folder created inside a stickybit-enabled folder could only be deleted by the creator itself. A nice implementation of sticky bit is the `/tmp` folder, where every user has write permission but only users who own a file can delete them. Remember files inside a folder which has write permission can be deleted even if the file doesn't have write permission. The sticky bit comes useful here.

2) As of SUID or SetUID bit, the executable which has the SUID set runs with the ownership of the program owner. That is, if you own an executable, and another person issues the executable, then it runs with your permission and not his. The default is that a program runs with the ownership of the person executing the binary.

3) The SGID bit is the same as of SUID, only the case is that it runs with the permission of the group. Another use is it can be set on folders, making nay files or folders created inside the SGID set folder to have a common group ownership.

```
bash-3.00$ chmod 4754 some_executable

bash-3.00$ ls -l

total 2

-rwsr-xr- 1 a435104 ccusers 50 Oct 17 05:28 some_executable
```

The extra "4? ahead of the permission set "754? specifies to always execute this file as the owner of the file.

The resulting permission has an "s" in place of "x". This is called setting the SUID/SGID/Sticky Bit.

Good example of the use of SUID bit is `/usr/bin/passwd`

Only root user has permission to modify the `/etc/passwd` file. If that's the case, how can a normal user change his password.

```
bash-3.00$ ls -l /etc/passwd

-rw-r-r- 1 root sys 6001 Aug 27 10:00 /etc/passwd
```

`/usr/bin/passwd` has it's SUID bit set. That means, irrespective of the user who is invoking the `passwd` program, the program always executes as the owner of the file (here root), granting it permission to modify `/etc/passwd` file.

```
bash-3.00$ ls -l /usr/bin/passwd
-r-sr-sr-x 1 root sys 27228 Aug 16 2007 /usr/bin/passwd
```

And what is SGID used for ? It is used when you want a program to execute always as a member of it's owners group.

```
bash-3.00$ chmod 2754 some_executable
bash-3.00$ ls -l
total 2
-rwxr-sr- 1 a435104 ccusers 50 Oct 17 05:28 test.sh
```

* `4000 (chmod u+s)` is suid; for files execute as owning user (often `root`).

* `2000 (chmod g+s)` is sgid; for files execute as owning group (often root); for directories the group on newly created files will be set to the directory's group rather than the creator's group. Typically used for shared directories.

* suid and sgid are ignored on scripts, due to the security risk

* `1000 (chmod +t)` is sticky bit ("save text image"); for files it used to be 'pin in memory' but is now ignored; for directories only root, file owner and directory owner can delete a file (even if non-owners have directory write permissions). Typically used for `/tmp. –t––`

* capital letters when doing ls -al usually means the permissions have been set incorrectly eg `-r-S–– SUID` is set, but owner execute is not set. However (?check?) `-rw––T` means no update of "last modified time"; usually used for swap files (not very common nowadays - swap is usually a partition).

What are the permissions used when you create a new file or directory?

After all, every file and directory does have permissions so what is used for default values?

First of all, almost any system will disable all execute permissions when you create a new file. Making a new directory, however will leave the execute bits turned on, otherwise you'd not be able to change into the directory. So if nothing else is set, directories will have permissions `wrxwrxwrx` (or `0777`) and files `wr-wr-wr-` (or `0555`). But if you actually do try it out, it is more likely your new directories will have permission `0755` and files 0644. This is because of the so called `umask`. Most systems set the default `umask` to `022` but you could change your own `umask` with the `umask` command to another value. What the heck is an `umask`, anyway? We just heard about the numeric representation of the file permissions and how they can be expressed with the digits 4, 2 and 1. The value passes to `umask` is a filter specifying the bits which are to be dropped when creating files and directories. So if you have a look at figure 4, you can see the `umask 022` in action and also how the execute bit is automatically dropped for files. SUID, SGID and sticky bits are rather special and never considered for creating new files and directories, so they are dropped automatically (unless some special bits on the parent directory turn them on, that is). Starting out with the maximum possible of the normal permission bits, we let them fall onto the filter. At the positions where the filter has an

entry, the permissions get caught and will not drop down to be applied to a new directory. When creating a new file, the remaining permissions are dropped onto another, automatic filter, snatching all execute permissions and letting pass through the rest. What falls through to the bottom will be the permissions being applied to a new file.