Les droits spéciaux sous GNU/Linux setuid setgid sticky-bit et umask

Quand on commence d'utiliser un système GNU/Linux et plus particulièrement la ligne de commande (what else), un sujet généralement très vite. Il s'agit des permissions sur les fichiers et les répertoires, ainsi que leur écriture octale quelque peu cryptique quand on s'y heurte les premières fois. Juste pour rappel, car cet article ne porte pas sur les permissions en général, voici les règles régissant les permissions sur un système GNU/Linux. Enfin nous terminerons cet article par une allusion à **UMASK**.

Les permissions sont déclinées en 3 catégories. Les droits du propriétaire, les droits du groupe et les droits des autres. Quand on fait un ls -l dans un répertoire, on obtient une sortie telle que :

```
drwxr-xr-x 3 root root 4096 Jun 23 2017 acpi/
-rw-r--r-- 1 root root 2981 Nov 4 2015 adduser.conf
-rw-r--r-- 1 root root 44 Nov 4 2015 adjtime
-rw-r--r-- 1 root root 227 Nov 5 2015 aliases
-rw-r--r-- 1 root root 12288 Mar 11 09:55 aliases.db
drwxr-xr-x 2 root root 4096 Nov 13 10:49 alternatives/
```

Prenons la ligne **drwxr-xr-x 3 root root 4096 Jun 23 2017 acpi/.** Le *d* en début de ligne est pour *directory* et indique donc qu'*acpi* est un répertoire. Ensuite vient la définition des permissions pour le propriétaire (ici root) *rwx* puis les droits du groupe (ici root) *r-x* et enfin les droits des autres *r-x.* Petite explication avant d'en venir au sujet principal.

```
 r veut dire read et accorde le droit de lecture notation octale : 4
w veut dire write et accorde les droits d'écriture/modification notation
octale : 2
x veut dire exécution pour un fichier exécutable (script ou binaire) mais
accorde également les droits d'entrer dans un répertoire notation octale :
1
```

Pour modifier les permissions sur un fichier ou un répertoire, on utilise la commande chmod. On peut procéder de 2 façons différentes:
Disons que nous souhaitons supprimer les droits d'accès au répertoire acpi aux autres. On pourrait alors faire :

```
 chmod o-x acpi o étant pour others (autres en anglais) et -x supprimant le
droit d'accès

soit

chmod 754 acpi 7 pour le propriétaire étant la somme de read write execute
ou enter soit (4+2+1) 5 pour le groupe étant la somme de read execute ou
enter (4+1) et 4 pour  les autres lecture seulement
```

La deuxième méthode ayant très largement ma préférence. Pour la première il est désormais facile de deviner les autres possibilités, chmod u pour users (le propriétaire) et g pour le

groupe. On peut bien sûr mixer : *chmod ug+x acpi*. Ce qui revient à accorder le droit d'entrer dans le répertoire pour le propriétaire et pour le groupe.

Venons en maintenant au sujet principal de cet article, les droits spéciaux. Parce-qu'en effet en plus des droits disons basiques évoqués ci-dessus, il existe 3 autres types de « droits » sur un système Unix/Linux. Il s'agit de *setuid*, *setgid* et ce qu'on nomme le s*ticky bit*.

- **SETUID**

Ce premier « droit » spécial ne s'applique qu'à des fichiers et pas des répertoires. Son utilisation est la plus simple à appréhender. Il faut d'abord comprendre le fait que quand un utilisateur exécute un script ou un lance un binaire quelconque, celui-ci se lance avec les droits de cet utilisateur. Par conséquent, et en fonction de ce que le script fait, il se peut que certaines actions ne soient pas permises avec le niveau de droits de l'utilisateur (par exemple écrire dans des fichiers appartenant à root). L'exemple le plus courant est l'utilisation de la commande *passwd.* Lorsqu'on souhaite changer son mot de passe, c'est cette commande qu'on utilise. Mais pour que le changement de mot de passe soit effectif, il faut que *passwd* puisse écrire dans les fichiers. */etc/passwd* et /*etc/shadow*. Problème, en tant qu'utilisateur nous n'avons pas le droit d'écrire dans ces fichiers. C'est là que **SETUID** entre en action. On dit alors que *passwd* est « *setuidé* » Celà veut dire que *passwd* est lancé avec les droits de *root* et du coup en tant qu'utilisateur l'écriture dans les fichiers précédemment cités est donc possible. Voici comment apparaît le **SETUID** :

```
-rwsr-xr-x 1 root root 59640 Sep 27 2017 /usr/bin/passwd*
```

Notez le s à la place du x pour le groupe de permissions du propriétaire (ici root). Sa valeur octale est 4. Les droits de *passwd* sont donc 4755

Pour positionner le **SETUID** il existe 2 méthodes, identiques à celles que nous avons vues au-dessus :

chmod 4755 qui donne les droits vus sur passwd

soit

chmod u+s

- **SETGID**

Contrairement à **SETUID**, **SETGID**, s'applique aussi bien aux fichiers qu'aux répertoires. Pour les fichiers, l'application et le comportement sont les mêmes. A savoir qu'un utilisateur qui lancerait un script ou un binaire, lui appartenant mais qui serait « *setguidé* » s'exécuterait avec les droits du groupe auquel il appartient. Ainsi un script « *setgidé* » « *root* » par exemple, pourrait écrire/lire dans des fichiers ou répertoires inaccessibles à l'utilisateur en temps normal.

En revanche quand un répertoire est «*setgidé* », le comportement observé change. On ne parle alors plus de droits d'exécution mais d'appartenance. En effet, tous les fichiers ou sous-répertoires qui seraient créés dans un tel répertoire, appartiendraient automatiquement au groupe auquel appartient le dossier. Si plusieurs utilisateurs peuvent et/ou doivent travailler

dans un même répertoire par exemple, on peut positionner le droit **SETGID** sur ce répertoire afin que tous les utilisateurs puissent accéder à son contenu sans restrictions liées au propriétaire qui a créé le fichier ou le sous-répertoire. C'est une solution bien plus élégante que de faire un horrible chmod 777 ! Voici comment apparaît le **SETGID :**

```
drwxr-sr-x 2 julien julien 4096 Jun 3 13:03 test/
```

Notez le s à la place du x pour le groupe de permissions du groupe (ici julien). Sa valeur octale est 2. Les droits de *test* sont donc 2755

Pour positionner le **SETGID** il existe 2 méthodes, identiques à celles que nous avons vues au-dessus :

chmod 2755 qui donne les droits vus sur test/

soit

chmod g+s

- **STICKY BIT**

Tout comme **SETGID**, le **STICKY BIT** peut s'appliquer aussi bien aux fichiers (et donc aux binaires/scripts) qu'aux répertoires. Lorsqu'il est positionné sur un fichier, celà permet à l'exéctuble de rester en mémoire même si le programme qui l'invoque est terminé, (d'où le terme sticky pour collant en anglais). Cette fonctionnalité était très utilisée sur les systèmes Unix dans les années 70/80 car il fallait du temps à l'époque pour charger un exécutable en mémoire et le rendre disponible à l'utilisation. Du coup positionner le **STICKY BIT** permettait au système de garder cet exécutable en swap pour le charger plus rapidement en mémoire par la suite. Avec la puissance des ordinateurs actuels et les vitesses de bus mémoire, cette méthode n'est quasiment plus utilisée.

En revance pour les répertoires il est toujours très utile. Postionné sur un dossier, le **STICKY BIT** permet d'interdire la suppression du répertoire et/ou de son contenu par quiconque n'en est pas propriétaire. L'exemple le plus parlant pour démontrer celà est le cas du répertoire */tmp.* N'importe quel utilisateur du système peut créer ou supprimer des fichiers ou des dossiers dans */tmp,* mais il le pourra pas supprimer ceux créés par quelqu'un d'autre. Ceci est permis grâce au positionnement du **STICKY BIT.** Voici comment il apparaît :

```
drwxrwxrwt 28 root root 12288 Jun 4 10:16 tmp/
```

Notez le t à la fin à la place du x pour le groupe de permissions des autres. Sa valeur octale est 1. Les droits de */tmp* sont donc 1777

A noter que si les droits d'exécution sont positionnés le t devient T.

Pour positionner le **STICKY BIT** il existe 2 méthodes, identiques à celles que nous avons vues au-dessus :

chmod 1777 qui donne les droits vus sur test/

soit

chmod o+t

Voilà pour l'explication des droits spéciaux sous GNU/Linux. Je vais terminer cet article par une allustion à **UMASK**.

UMASK (User file creation mode **M**ask) permet de définir les droits par défaut qui seront appliqués sur la création des fichiers et des répertoires. Il fonctionne de manière disons « inversée » car il ne va pas ajouter des droits mais en enlever. Je m'explique.

Par définition sur un système GNU/Linux, les fichiers sont créés avec les droits 666 (rw-rw-rw-) qui donnent les droits de lecture et d'écriture à tout le monde. Les répertoires sont eux créés avec les droits 777 (rwxrwxrwx) qui donnent tous les droits à tout le monde. La valeur **UMASK** par défaut est de 022. Ce qui veut dire qu'au final un fichier sera créé avec les droits 644 (rw- r– r–) et que par conséquent les droits d'écriture seront supprimés pour le groupe et les autres soit 666 – 022 (2 correspondant au droit d'écriture comme vu en introduction). Un répertoire quant à lui sera créé avec les droits 755 (rwx-r-x-r-x). Ici aussi le droit d'écriture est bien supprimé pour le groupe et les autres.

Il est possible de modifier les valeurs **UMASK** par défaut soit de manière temporaire en exécutant simplement :

```
umask 077
```

Ou en allant modidier directement la valeur du système dans le fichier

```
/etc/login.defs
```