Fichiers, Répertoires et droits d'accès

1 Droits d'accès à un fichier

Utilisez `man chmod` comme aide-mémoire.

Exécutez `ls -l /etc/shadow` . Qui a le droit de faire quoi avec ce fichier ? Pourquoi ?

Mêmes questions pour `/usr/bin/passwd` (qui est censé vous permettre de changer de mot de passe). Notez la présence de `s`, le bit SETUID, pourquoi est-il nécessaire ici ?

Mêmes questions pour les fichiers spéciaux `/dev/null` (la «poubelle»), `/dev/mem` (la mémoire physique), `/dev/pts/*` (les terminaux) et le programme `/usr/bin/bsd-write` (notez notamment comment interagissent ces deux derniers).

Compilez le programme suivant (il ne fait qu'attendre qu'on le tue avec `control-C` ou `kill`) :

```
int main(void) { pause(); }
```

En binômes, jouez avec le bit SETUID : copiez le programme compilé dans /tmp (pas dans votre home, car le fonctionnement SETUID y a été désactivé), appliquez le bit SETUID à la copie (`chmod u+s fichier`), et vérifiez que si quelqu'un d'autre le lance, l'USER apparaissant dans `ps axuf` est le vôtre. Qui peut le tuer ? (essayez !). Note : pour lancer un shell sous une autre identité, vous pouvez utiliser `su - lelogin`

Lisez `man getuid` Ajoutez au programme l'affichage de l'UID réel et de l'UID effectif (tapez `id` pour savoir le vôtre). Lisez `man setuid`, vérifiez que vous pouvez remettre l'UID réel dans l'UID effectif.

À quoi cela peut-il servir ?

Lisez `man setresuid` , à quoi peut servir d'UID sauvé ?

Imaginez qu'un jour vous trouviez la machine avec un shell root lancé. Comment faire pour à l'avenir pouvoir très facilement devenir root sur cette machine (sans que les administrateurs s'en rendent compte) ? Pourquoi le fonctionnement SETUID a-t-il été désactivé dans les homes ?

2 Groupe

Dans la sortie de `id`, vérifiez à quel groupe vous appartenez (GID). Vérifiez que les fichiers vous appartenant ont effectivement ce GID. Créez un fichier que seul quelqu'un du groupe pourra lire. Pouvez-vous vérifier vous-même que cela marche ? À quoi cela peut-il servir ? Mêmes questions pour le droit en écriture.

À quoi pourrait servir SETGID ? (`chmod g+s`)

3 Droits d'accès à un répertoire

Créez un répertoire bidon, et avec `chmod`, vérifiez la différence subtile entre droit d'exécution et droit de lecture (entre `cd` et `ls` par exemple)

En créant quelques répertoires, comment faire pour que seule une personne de votre groupe connaissant un certain mot de passe peut arriver à rentrer dans un répertoire et y écrire des fichiers ? Vérifiez avec votre voisin.

Lancez `ls -ld ~sthibault/test` Il y a un bit 't' qui est activé, lisez sa description dans `man chmod`. Essayez d'y créer un fichier. Pouvez-vous relire ce fichier ? Pouvez-vous le modifier ? Pouvez-vous le supprimer ? Pouvez-vous voir qu'il y a un fichier toto stocké dans ce même répertoire ? Pouvez-vous le supprimer ? Pouvez-vous supprimer le fichier créé par votre voisin ?

Observez que `/tmp` a également le bit 't' positionné, pourquoi ?

4 Création de fichiers « sensibles »

Par défaut, lorsque vous créez des fichiers, ils sont lisibles par tout le monde (`rw-r--r--`). Pour se prémunir de cela, on pourrait penser effectuer :

1. `touch fichier` (pour créer le fichier)
2. `chmod g-r,o-r fichier` (pour empêcher la lecture)
3. `echo foo >> fichier` (pour ajouter des données)

Mais entre la première et la deuxième étape, une autre personne (connectée par `ssh` par exemple) pourrait ouvrir le fichier. Essayez en binôme **dans /tmp sur une même machine** (dans un montage réseau cela ne fonctionne pas) :

Bob	Alice
`touch /tmp/fichier`	
	`tail -f /tmp/fichier` (laissez-le tourner)
`chmod g-r,o-r /tmp/fichier`	
`echo foo >> /tmp/fichier`	

`foo` apparaît sur l'écran d'Alice ! Même si Bob essayait de supprimer `fichier`, le processus `tail` d'Alice pourrait continuer à le parcourir.

Pour créer des fichiers avec des permissions restreintes de manière sûre, il faut donc utiliser `umask` : essayez

```
umask 077
touch unautrefichier
ls -l unautrefichier
```

5 ACL

Il peut être utile de permettre à un ensemble de personne d'avoir certaines permissions, un autre ensemble de personnes d'avoir d'autres permissions, etc... Les droits Unix standards ne suffisent pas pour cela, il faut utiliser les ACL (Access Control List).

À l'ENSEIRB, les ACL sont par exemple disponibles sur la machine `travail32` , dans le répertoire `/tmp` , connectez-vous donc à cette machine et déplacez-vous dans ce répertoire.

Créez-y un fichier à votre nom (désigné ici par `toto`). Lancez la commande `getfacl toto` on reconnait là les droits Unix standards. Travaillez en binôme, dont on note ici `login1` et `login2` les logins respectifs. `login1` peut lancer `setfacl -m user:login2:rw toto` pour donner le droit d'écriture au fichier à `login2` . Vérifiez que c'est bien le cas. On peut ainsi donner des droits à n'importe quelle liste de logins. On peut faire de même pour des groupes Unix.

6 setuid et scripts shell

Essayez en binôme de créer et faire fonctionner un script shell setuid dans `/tmp`. Constatez que le bit setuid n'a aucun effet pour un script shell.

La raison est que lorsque l'on exécute un script shell `foo`, le noyau exécute en fait d'abord `/bin/sh foo`, et le shell se débrouille pour ouvrir le fichier `foo` et interpréter les commandes contenues. Si le bit setuid avait effet, le shell obtiendrait l'identité supplémentaire avant même d'avoir ouvert le fichier `foo`. Une attaque possible serait alors de créer un lien `/tmp/bar` pointant vers `foo`, exécuter le script à l'aide de ce lien `/tmp/bar` (le noyau exécute alors `/bin/sh /tmp/bar`), et immédiatement faire pointer le lien symbolique vers un autre script (avant que le shell ne l'ouvre).

Comment pouvoir tout de même avoir l'effet setuid sur certains scripts shells dont le chemin serait connu ? Implémentez-le.