Cours Linux et applications Open-Source

Module n° 400

Sécurisation et droits

21 février 2002

Frédéric Schütz

schutz@mathgen.ch

ESNIG

2001–2002

Informations supplémentaires

La dernière version de ce texte est disponible sur le web à l'adresse

http://www.mathgen.ch/cours/droits/

En addition à la licence ci-dessus, l'auteur original vous serait reconnaissant de lui envoyer les modifications que vous apporterez à ce texte ou toute autre suggestion, afin qu'il puisse les prendre en compte dans une prochaine version et améliorer ce document.

Si ce document est utilisé dans un but lucratif, il n'est pas interdit d'en faire profiter l'auteur original ;-) (ou une organisation à but non-lucratif choisie avec lui)

Toute question, modification, commentaire ou autre peut-être envoyé à l'adresse électronique schutz@mathgen.ch, ou par courrier :

Frédéric Schütz
Ch. de Mont-Rose 33
1294 Genthod
Suisse

Table des matières

Security is a process, not a product.
— Bruce Schneier

1 Utilisateurs et groupes

Sur un système Unix, tout utilisateur est référencé sous forme d'un nombre, le UserID (UID), avec lequel sont faites toutes les relations entre un fichier ou un processus et son propriétaire. Le système permet également, pour simplifier les choses, d'y associer un nom en clair à l'aide du fichier `passwd` que l'on verra plus loin. Il est possible de créer plusieurs utilisateurs ayant le même UID, mais cela est généralement fortement déconseillé.

Chaque utilisateur fait en plus partie d'un groupe principal, également référencé par un nombre, le Group ID (GID), et optionnellement d'autres groupes. Un même nombre peut référencer à la fois un groupe et un utilisateur, sans qu'il n'y ait de relation entre eux et plusieurs personnes peuvent appartenir au même groupe. La correspondance entre GID et nom du groupe, ainsi que la liste des gens qui appartiennent à un groupe (excepté leur groupe principal) sont stockées dans le fichier `/etc/group`.

La commande `id` permet de connaître vos UID et GID, les groupes auxquels vous appartenez, ainsi que les noms qui leurs sont associés :

```
[schutz]$ id
uid=1211(schutz) gid=350(telecom) groups=350(telecom),1105(internet),900(tcs),
1115(mathinfo),1121(asstsc1)
```

L'attribution des groupes par défaut pour les nouveaux utilisateurs varie selon les distributions Linux. Certaines définissent un groupe `users` auquel appartiennent tous les nouveaux utilisateurs (ce qui diminue l'utilité de la notion de groupe, en identifiant les notions de "groupe" et d' "autres"). D'autres, comme Debian, créent un nouveau groupe par utilisateur, avec le même nom et le même ID. Cette solution est plus sûre par défaut (se tromper dans les droits accordés au groupe est moins important que dans la précédente), et elle peut être pratique dans certains cas (voir les travaux pratiques). Dans l'exemple de la commande `id` ci-dessus, le système de groupes est plus évolué, avec des groupes qui correspondent à de réels groupes de personnes.

Debian définit également quelques groupes supplémentaires pour faciliter le partage de certaines tâches. Ainsi, il suffit qu'un utilisateur appartienne au groupe `src` pour qu'il puisse travailler avec les fichiers sources du système, dans le répertoire `/usr/src`, sans avoir besoin de droits supplémentaires.

Un UID de 0 est considéré par le système comme un superutilisateur (l'administrateur système) et possède tous les droits. Il est toujours appelé `root`, mais c'est une convention plus qu'une obligation : seul son UID détermine ses privilèges.

Le superutilisateur dispose de deux commandes pour changer le propriétaire d'un fichier : `chown nouveau-proprietaire fichier` permet de changer l'utilisateur propriétaire, tandis que `chgrp nouveau-groupe fichier` permet de changer son groupe.

Ces considérations ont leur importance même si vous êtes la seule personne à utiliser votre ordinateur, car il est indispensable que vous ne travailliez pas sous le nom d'utilisateur `root` quand vous n'avez pas spécifiquement besoin de droits privilégiés. Il faut donc qu'il y ait au moins deux utilisateurs définis, `root` et vous-même sous un autre nom. Bien entendu, ces deux utilisateurs correspondent à une seule personne physique, mais ils vous permettent de travailler sur vos fichiers, sans craindre de mettre en péril le système lui-même en faisant une fausse manœuvre. Vous n'utiliserez `root` que quand ce sera vraiment nécessaire.

1.1 Protection de l'utilisateur `root`

De par son importance et les privilèges qu'il possède, les possibilités d'accès de l'utilisateur `root` sont à traiter de façon particulièrement prudente.

Certains proposent de créer un deuxième superutilisateur avec un nom différent (mais le même UID 0, évidemment), et de ne jamais utiliser `root`. De cette manière, on verrait immédiatement dans les logs si quelqu'un d'autre a obtenu cet accès. Avoir plusieurs utilisateurs avec le même UID n'est pas une très bonne idée, et d'autres solutions, meilleures, peuvent être mises en place.

Pour commencer, il faut limiter au maximum (voire supprimer) les possibilités de login direct de `root`. Le fichier `/etc/securetty` liste les terminaux sur lesquels `root` peut se loguer (les terminaux considérés comme sûrs) : dedans ne devraient figurer que des lignes `tty`*x* où *x* est un nombre entre 1 et 12 (dépendant de la configuration), ce qui n'autorise l'accès à `root` que depuis les terminaux locaux de la machine. Vous pouvez également interdire à `root` de se connecter via `ftp` en ajoutant son nom dans le fichier `/etc/ftpusers` (cette interdiction est souvent déjà configurée).

Un utilisateur non-local ne peut plus faire de login `root`, mais il peut utiliser la commande `su` pour le devenir après s'être logué sous son propre nom. Cette commande a l'avantage d'enregistrer dans le fichier log le nom de l'utilisateur qui l'a utilisée. Pour plus de sécurité, on peut restreindre son accès à un groupe d'utilisateurs de confiance (voir les problèmes pratiques).

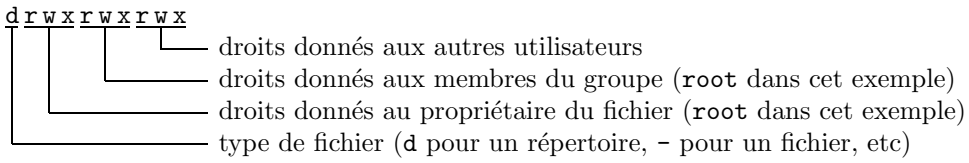
Bien évidemment, ces mesures de sécurité sont inutiles si l'utilisateur à distance se connecte à l'aide d'un système non-sûr (par exemple `telnet`) et que le mot de passe de `root` est ensuite accessible en clair à toute personne connectée au réseau ! Comme toujours, la sécurité forme une longue chaîne, et un maillon faible suffit à la faire sauter entièrement.

2 Droits d'accès Unix

Chaque fichier ou répertoire sous Unix possède une série de permissions qui décrivent qui a accès à ce fichier, et sous quelles conditions. La liste de ces permissions est accessible en utilisant la commande `ls -l`, le résultat étant quelque chose du genre :

```
-rw-------    1 root     root          671 Oct 10 20:12 shadow
-rw-r--r--    1 root     root          166 Oct 10 21:33 shells
drwxr-xr-x    2 root     root         4096 Jun 20 19:17 site-start.d/
drwxr-xr-x    2 root     root         4096 Oct 10 20:12 skel/
```

Les permissions sont inscrites dans le premier bloc d'informations, qui est composé de 4 parties :



Le fonctionnement des permissions est différent selon qu'on l'applique à des fichiers ou à des répertoires, malgré qu'elles soient codées avec les mêmes lettres `r`, `w` et `x`. D'autres droits plus spécifiques existent, que l'on verra séparément.

2.1 Les droits d'accès pour les fichiers

Pour les fichiers, les droits d'accès ont la signification suivante :

`r` ("read", lecture) signifie que vous avez le droit de lire le contenu du fichier (par exemple en faisant `cat fichier`), ou de le copier.

`w` ("write", écriture) signifie que vous pouvez modifier le contenu du fichier (que ce soit pour ajouter ou enlever quelque chose), ou l'écraser avec le contenu d'un autre fichier. Vous pouvez le vider complétement jusqu'à ce qu'il ait une taille nulle, mais cet attribut ne vous donne pas le droit de l'effacer (qui dépend du répertoire).

`x` ("execution") signifie que le fichier est un programme exécutable que vous avez le droit de lancer en tapant son nom.

La plupart du temps, le bit `x` a surtout un intérêt informatif — différencier le fichier suivant qu'il contienne un programme ou des données — d'autant plus qu'une personne qui aurait le droit de lecture, mais pas celui d'exécution pourrait toujours copier le fichier, changer ses permissions puis l'exécuter (mais le nom et l'emplacement du fichier exécuté serait différent). A l'inverse, on peut vouloir donner à un utilisateur les droits d'exécution, mais pas ceux de lecture, ce qui l'empêchera d'examiner le programme ou de le copier.

On peut noter que les liens symboliques ne sont pas concernés par ces permissions, et que ce sont uniquement les permissions du fichier pointé qui s'appliquent. La seule opération que l'on peut faire sur un lien, l'effacer (et éventuellement le remplacer par un autre ensuite), dépend seulement des permissions du répertoire.

2.2 Les droits d'accès pour les répertoires

Les droits d'accès aux répertoires ont la signification suivante :

`r` signifie que vous avez le droit de voir la liste des fichiers du répertoire (mais pas leurs attributs ou leur taille).

`w` signifie que vous avez le droit d'ajouter, d'effacer ou de renommer des fichiers.

`x` ("access") signifie que vous avez le droit d'accéder à ce répertoire, lors d'une recherche par exemple. Vous pouvez faire de ce répertoire votre répertoire courant (par exemple à l'aide de la commande `cd`) et ouvrir des fichiers à l'intérieur de ce répertoire.

Si vous avez la permission `x`, mais pas la permission `r`, vous ne pouvez pas obtenir la liste des fichiers du répertoire. Cependant, si vous connaissez le nom d'un de ces fichiers, et avez les droits nécessaires dessus, vous pouvez l'ouvrir ou l'exécuter.

2.3 Les bits SUID, SGID et Sticky

Toute commande qui est exécutée hérite normalement des droits de la personne ou du processus qui l'a lancée, indépendamment de ceux du propriétaire du fichier. Il peut néanmoins arriver qu'un utilisateur doive accomplir une tâche qui va au delà de ses droits : par exemple, pour changer votre propre mot de passe, vous devez modifier le fichier `/etc/passwd` — mais vous n'avez évidemment pas le droit d'écrire directement dans ce fichier.

Pour résoudre cela, Unix utilise des permissions supplémentaires, qui autorisent un programme à s'exécuter avec les droits du propriétaire du fichier, `root` la plupart du temps, plutôt qu'avec ceux de l'utilisateur qui l'exécute. Ce système doit être utilisé de façon très prudente, car le programme exécuté héritera de *toutes* les permissions du propriétaire, et vous devrez être certain qu'il n'accomplira que les tâches pour lesquelles il est prévu. Par exemple, il ne faut pas que le programme `passwd` permette à un utilisateur de modifier le mot de passe de quelqu'un d'autre !

Cette permission peut s'appliquer au niveau du propriétaire ; on parle alors de bit SUID (Set User ID), et le programme reçoit alors l'identité (et donc les privilèges) de son propriétaire lors de son exécution. Il peut également s'appliquer au niveau du groupe, c'est alors le bit SGID (Set Group ID), et le programme reçoit alors les privilèges de son groupe.

Au niveau des permissions, ces permissions se traduisent par un `s` qui remplace le `x` de l'autorisation d'exécution (si le programme ne possédait pas la permission `x`, la nouvelle permission est marquée par un `S` majuscule) :

```
-rwsr-xr-x    1 root     root        25692 Oct 10 21:33 /usr/bin/passwd*
```

Au moment de l'exécution, l'UID du processus ne change pas, mais une autre variable, l'Effective Used ID (EUID) (qui est normalement égale à l'UID) prend l'UID du propriétaire du fichier. Le fait de garder intacte la valeur de l'UID permet au programme de se rappeler de qui l'a lancé. Cela peut être utile pour que `passwd` se rappelle quel mot de passe il est censé changer, mais surtout, certains programmes (par exemple `bash`) sont conçus pour refuser de fonctionner avec le bit SUID, et abandonnent leurs privilèges extraordinaires s'ils constatent que UID≠EUID.

Il n'est pas possible d'utiliser les bits SUID sur des shell-scripts, ceux-ci étant considérés comme trop dangereux. La commande `sudo [options] commande` permet néanmoins d'exécuter des scripts avec les privilèges de `root` ou d'un autre utilisateur. La liste des utilisateurs autorisés à utiliser cette commande est contenue dans le fichier `/etc/sudoers`, et l'utilisateur doit entrer son propre mot de passe au moment de l'exécution.

La permission SGID peut également être utilisée pour les répertoires, avec une signification totalement différente. N'importe quel fichier créé dans un tel répertoire fera automatiquement partie du même groupe que le répertoire. Ceci est utile par exemple si plusieurs utilisateurs partagent un répertoire, et que l'on veut s'assurer que les fichiers qu'ils créeront auront le bon groupe afin qu'ils soient accessibles aux autres (vous pouvez aussi choisir manuellement le groupe par défaut qu'auront vos nouveaux fichiers en faisant `newgrp` *`groupe`* — vous devez bien entendu faire partie du groupe).

Il existe encore une autre permission spéciale, le sticky bit (avec comme lettre correspondante `t` dans les permissions des autres utilisateurs). Celui-ci est surtout utilisé pour les répertoires qui sont autorisés en écriture à tous le monde, tel que `/tmp` :

```
drwxrwxrwt    4 root     root         4096 Nov 17 14:44 tmp/
```

Dans un tel répertoire, n'importe quel utilisateur peut créer un fichier, mais seul le propriétaire du fichier ou du répertoire peut l'effacer. Historiquement, le sticky bit était aussi utilisé pour les fichiers, mais il n'a plus aucun effet sous Linux.

2.4 Changer les droits d'accès

La commande `chmod` permet de changer les droits d'accès à un fichier. Elle admet deux syntaxes différentes :

- **Le mode symbolique**, où la commande est de la forme

  ```
  chmod [options] [augo][+-=][rwxst] fichiers
  ```

  et les permissions spécifiées comme ceci :
  - `[agou]` spécifie les permissions de qui sont modifiées : `a` pour modifier celles de tout le monde, `u` pour modifier celles du propriétaire du fichier, `g` pour modifier celles du groupe, et `o` pour modifier celles des autres utilisateurs.
  - `[+-=]` spécifie ce qui est fait des permissions données : `+` signifie "ajouter la permission", `-` signifie l'enlever, et `=` signifie remplacer les permissions existantes par les nouvelles.
  - `[rwxst]` spécifie quelles sont les permissions qui doivent être changées.

  D'autres permissions et options, moins courantes, peuvent être utilisées, et sont décrites en détails dans les pages de manuel. Il est possible de spécifier plusieurs permissions à la fois en les séparant par des virgules, par exemple `chmod u+rw,u-x,g+r,o-rwx fichier`.
- **Le mode octal** : chaque ensemble de permissions peut être représenté comme un nombre de quatre chiffres en base 8, appelé le *mode*, qui est calculé en additionnant les nombres correspondant à chacune des permissions accordée, selon le tableau suivant :

Nombre	Permissions
4000	Set User ID à l'exécution (SUID)
2000	Set Group ID à l'exécution (SGID)
1000	Sticky Bit
0400	Lecture par le propriétaire
0200	Ecriture par le propriétaire
0100	Exécution par le propriétaire
0040	Lecture par le groupe
0020	Ecriture par le groupe
0010	Exécution par le groupe
0004	Lecture par les autres
0002	Ecriture par les autres
0001	Exécution par les autres

Ainsi, un fichier qui a les permissions `-rwxr-x---` aura un mode de 0750 (ou 750), correspondant à l'addition de 0400, 0200, 0100 (lecture, écriture et exécution par le propriétaire), 0040 et 0010 (lecture et exécution par le groupe). Ce mode est souvent utilisé pour rechercher les fichiers qui ont certaines permissions (avec la commande `find`), et avec la deuxième syntaxe de la commande `chmod` (qui ne tient aucun compte des permissions antérieures) :

```
chmod [options] mode fichiers
```

2.5 Umask

Beaucoup d'utilisateurs ne se posent pas la question de savoir quels sont les permissions accordées par défaut à un fichier nouvellement créé — sera-t-il accessible à tout le monde en lecture, ou au contraire restreint au seul propriétaire ?

Le mode de création de fichier par défaut est précisé par la commande `umask mode`, qui est une commande interne du shell. Le mode est une valeur en octal spécifiant les droits que vous ne voulez *pas* donner à un nouveau fichier. Ainsi, un umask de `022` signifie que vous ne voulez pas donner les droits d'écriture ni au groupe, ni à tout le monde.

Si aucun `umask` n'est précisé, la plupart des systèmes Unix créent les nouveaux fichiers avec un mode `666` (et `777` pour les répertoires), les rendant accessibles en lecture et en écriture à tout le monde ! Pour cette raison, l'administrateur système spécifie *toujours* une valeur plus restrictive dans le fichier `/etc/profile`, qui est exécuté lors du login de chaque utilisateur, libre à celui-ci de changer ensuite cette valeur. La commande `umask` sans argument permet de connaître la valeur actuelle du masque. Les valeurs les plus courantes sont `022`, suivi de `027` et `077`, qui interdit tout accès à d'autres personnes que l'utilisateur. Si les utilisateurs connaissent le système de droits Unix, un umask de `022` peut être adapté, mais en général, `077` peut être un meilleur choix, leur évitant de laisser des accès ouverts par ignorance.

Les fichiers ajoutés par FTP n'utilisent pas l'umask de l'utilisateur et les droits de création par défaut doivent être configurés séparément. Avec `ftpd`, l'option `-u umask` permet de les spécifier au lancement, tandis qu'avec `proftpd`, l'option se trouve dans le fichier `/etc/proftpd.conf`.

2.6 Recherche des permissions dangereuses

Pour garantir la sécurité d'un système, il est indispensable de vérifier régulièrement si les fichiers sensibles ont les bonnes permissions, et les corriger dans le cas contraire. La commande `find`, avec ses multiples options, est l'outil idéal pour mener ce genre de recherche.

2.6.1 Fichiers SUID/SGID

Tout programme qui s'exécute avec la permission SUID `root` permet à un utilisateur de réaliser des tâches avec les privilèges du superutilisateur, qu'il n'a pas en temps normal. La plupart du temps, un tel programme ne permet de réaliser que les tâches pour lesquelles il est prévu. Mais il reste toujours possible qu'un pirate trouve le moyen de forcer un tel programme à faire autre chose, comme cela peut arriver à la suite des attaques de type "buffer-overflow" qui permettent de faire exécuter un bout de code arbitraire à un programme, et tout programme SUID `root` peut potentiellement subir une attaque de ce genre, qui permettra au morceau de code malicieux de s'exécuter avec les privilèges de `root`. Pour cette raison, il est indispensable de limiter au maximum les programmes auquels on donne cette permission. Plusieurs méthodes sont à mettre en œuvre.

Tout d'abord, il n'y a généralement pas de raison pour que les utilisateurs attribuent eux-mêmes des permissions SUID à leurs fichiers, ceci n'étant utile que pour `root`. La commande `mount` possède une option `nosuid` qui rend inopérantes de telles permissions sur un système de fichiers, et il est conseillé de l'activer pour tous les systèmes de fichiers sur lesquels des utilisateurs autres que `root` ont accès en écriture. Par exemple, pour empêcher les utilisateurs de créer de tels fichiers dans leur répertoire personnel, on peut modifier le fichier `fstab` (qui liste les partitions qui seront montées automatiquement au démarrage) de la façon suivante :

```
/dev/hda7 /home ext2 rw,nosuid                    0       2
```

Il est également nécessaire de recenser tous les fichiers de votre système qui ont cette autorisation, avec une commande telle que

```
find / -type f -perm +6000 -ls
```

Pour chacun des fichiers ainsi trouvé, il faut examiner si l'accès à ce fichier est vraiment nécessaire pour vos utilisateurs, et si non, désactiver les permissions SUID correspondantes. Par exemple, l'accès à des utilitaires tels que `ping` et `traceroute`, qui ont déjà causé plusieurs problèmes de sécurité, est superflu la plupart du temps. Si nécessaire, vous pouvez toujours jouer sur les permissions de groupe pour donner un accès à certaines personnes seulement (voir les problèmes pratiques).

La distribution Debian possède un script appelé `checksecurity` pour contrôler les permissions SUID des fichiers du système. Cette commande recherche les fichiers SUID sur tous les systèmes de fichiers montés, et indique toute différence qu'elle trouve par rapport à ceux qu'elle avait trouvés lors de la dernière exécution. Elle indique également les systèmes de fichiers qui sont montés de façon non-sûre (i.e. sans l'option `nosuid`).

2.6.2 Fichiers autorisés en écriture pour tout le monde

Très peu de fichiers ou répertoires sont censés être accessibles en écriture à tout le monde (`/tmp`, avec la permission Sticky Bit, est l'un d'eux), et ils peuvent poser d'importants problèmes de sécurité. On peut les rechercher avec la commande suivante :

```
find / -perm -2 \( -type f -o -type d \) -ls
```

La commande `find` ayant une syntaxe qui n'est pas toujours évidente, on peut lire celle-ci de la façon suivante :

`find`	cherche les fichiers
`/`	depuis la racine
`-perm -2`	qui ont au moins une permission "`w`" pour tout le monde (`002`)
`\( ... \)`	et
`-type f`	qui sont des fichiers
`-o`	ou
`-type d`	des répertoires
`-ls`	et affiche leurs informations complètes (`-print` n'afficherait que le nom).

On se restreint uniquement aux fichiers et aux répertoires, car comme on l'a vu, certains autres types de fichiers, tels que les liens symboliques, sont autorisés en écriture pour tout le monde sans que ce soit dangereux.

2.6.3 Fichiers n'appartenant à personne

Un fichier n'appartenant à personne est un fichier dont l'UID du propriétaire ne correspond à aucun utilisateur du système. Ceci peut arriver dans certains cas (par exemple, après effacement d'un utilisateur sans enlever tous ses fichiers, ou si vous accédez à une partition montée en réseau), mais c'est assez rare sinon, et c'est souvent l'indice d'une intrusion dans votre système. On peut rechercher de tels fichiers avec la commande

```
find / -nouser -o -nogroup -ls
```

2.7 Sous quel nom d'utilisateur doivent tourner les services ?

Nous avons parlé au début de ce cours du danger qu'il y a à utiliser `root` pour effectuer des tâches qui ne requièrent pas de privilège du superutilisateur.

Dans le même ordre d'idées, il faut éviter de faire tourner des services avec l'UID `root` s'il ne leur est pas nécessaire. C'est pour cette raison que le fichier `/etc/passwd` est rempli d'utilisateurs tels que `mail`, `postgres`, etc, qui permettent à ces programmes de tourner sous leur nom propre (sauf pour les petites parties qui requièrent des privilèges plus importants) et de limiter les problèmes de sécurité potentiels.

Certains programmes sont lancés avec les privilèges de `root`, sans que cela ne pose de problème de sécurité. Ainsi, le serveur web Apache, dans une situation normale, a besoin des privilèges du superutilisateur pour se lier au port 80 (l'accès à tous les ports en dessous de 1024 est réservé à `root`). Mais dès cette opération terminée, Apache abandonne l'identité et les privilèges de `root` pour tourner sous le nom d'utilisateur spécifié dans ses fichiers de configuration, typiquement `www-data`.

Dans le même ordre d'idées, le programme de cryptographie GPG (Gnu Privacy Guard) a besoin du bit SUID, mais les privilèges `root` ne sont gardés uniquement le temps de réserver de la mémoire privilégiée (i.e. qui ne sera jamais recopiée sur le cache du disque). Dès cette opération terminée, GPG redevient un programme normal tournant sous l'identité de celui qui l'a lancé.

2.8 Les permissions étendues de ext2fs

Le système de fichiers `ext2fs` généralement utilisé sous Linux permet d'attribuer des permissions supplémentaires aux fichiers qu'il contient. Deux commandes sont utilisées à cet effet (voir les pages de manuel pour le détail des options existantes) :

```
lsattr [options] fichiers
```

qui permet de voir les attributs des fichiers, et

```
chattr [options] [+-=][attributs] fichiers
```

pour les changer. On peut choisir entre `+` pour ajouter les attributs, `-` pour les enlever et `=` pour ne mettre que ceux qui sont spécifiés et annuler ceux qui étaient là antérieurement.

Parmi les différents attributs, on trouve :

`A` désigne un fichier dont l'atime (access time) n'est pas modifié. Ceci permet de gagner un peu de temps puisque le système n'est plus obligé de mettre à jour ce champ à chaque lecture du fichier.

`a` un fichier avec cet attribut ne peut être ouvert en écriture que pour ajouter du contenu, mais jamais pour en enlever ou le modifier. Seul l'administrateur système peut définir ou enlever cet attribut.

`d` désigne un fichier qui ne sera pas pris en compte par la commande `dump` lors d'une sauvegarde.

`i` désigne un fichier qui ne peut pas être modifié, effacé, ni renommé, et aucun lien ne peut être créé dessus. Seul l'administrateur système peut définir ou enlever cet attribut.

`S` si un fichier avec cet attribut est modifié, les changements sont faits immédiatement sur le disque (comme ils le seraient si le système de fichier était monté avec l'option `sync`).

D'autres attributs intéressants, comme la compression automatique d'un fichier par le noyau, sont prévus pour une version future. Plus étonnant, l'attribut `s`, qui oblige le système à recouvrir le fichier de valeurs zéro avant de l'effacer pour éviter qu'on puisse retrouver son contenu, était supporté par le noyau Linux jusqu'à la version 1.2, mais a disparu des versions suivantes, pour éviter de donner une impression de fausse sécurité aux utilisateurs.

Typiquement, on attribuera aux fichiers de logs l'attribut `a`, pour éviter leur effacement par un intrus, et l'attribut `i` à certains programmes importants tels que `/bin/login`. Ces attributs étendus étant indépendants du système de droits Unix standard, même `root` (ou quelqu'un qui aurait réussi à le devenir) ne peut pas effacer ou modifier des fichiers qui auraient l'attribut `a` ou `i`.

Il reste cependant un problème : `root` peut toujours enlever l'attribut protecteur, puis modifier le fichier (quitte à remettre l'attribut ensuite). La solution s'appelle "kernel capabilities" et a fait son apparition dans la version 2.2 du noyau Linux. Sans rentrer dans les détails, les capabilities permettent de séparer les privilèges de `root` en petites parties qui peuvent ensuite être attribuées aux processus qui en ont besoin, sans devoir faire du "tout ou rien". On peut trouver la liste des capabilities dans les sources du noyau, dans le fichier `/usr/src/linux/include/linux/capabilities.h` (qui peut être situé ailleurs suivant la distribution). La commande `lcap` (qui n'est pas disponible en standard, mais peut être trouvée sur la page `http://pw1.netcom.com/~spoon/lcap`) permet de supprimer définitivement des capabilities du système (il n'existe aucun moyen de les restaurer autre que de le redémarrer), même pour l'utilisateur `root`. Il faut donc qu'au démarrage du système, les deux lignes suivantes soient exécutées (depuis un fichier qui est lui-même protégé) :

```
lcap CAP_LINUX_IMMUTABLE
lcap CAP_SYS_RAWIO
```

La première ligne supprime la possibilité de changer les attributs `a` et `i`, même pour `root`, tandis que la deuxième empêche l'accès direct (brut) aux disques (via les fichiers `/dev/*`) qui pourrait être utilisé pour modifier directement les attributs sans passer par le système de fichiers. La seule manière de modifier des attributs étendus est alors de redémarrer le système en mode "single user", ce qui nécessite un accès physique à la machine.

On peut trouver plus d'informations sur le sujet sur les pages suivantes :

- `http://www.securityfocus.com/focus/linux/articles/ext2attr.html`
- `ftp://ftp.guardian.no/pub/free/linux/capabilities`
- `http://www.hsc.fr/ressources/presentations/linux2000/index.html.fr`

2.9 Access Control Lists

Le système standard de permissions UNIX a l'avantage d'être relativement simple, mais il permet difficilement d'établir des permissions fines. Si vous voulez donner accès à un fichier à quelques personnes, il vous faudra créer un groupe pour ces personnes et le tout devient rapidement ingérable si le nombre de telles combinaisons augmente.

Les ACL (Access Control Lists) ajoutent au système un mécanisme permettent de spécifier, pour chaque fichier, une liste de droits d'accès (les mêmes `r`, `w` et `x`) pour chaque utilisateur ou groupe.

Le système n'est pas installé par défaut, et il est nécessaire de recompiler le noyau Linux après son installation. Vous pourrez ensuite utiliser les commandes `getfacl` (GET File Access Control List) ou `setfacl` pour respectivement lire ou modifier ces attributs étendus. Les ACL peuvent être très utile, mais nécessitent une gestion assez rigoureuse (et sur la durée) pour être efficaces, ce qui explique qu'elles sont relativement peu utilisées. On peut trouver plus d'informations sur le site `http://acl.bestbits.at`.

2.10 Les droits d'accès en pratique

Nous avons détaillé ci-dessus plusieurs méthodes permettant de gérer plus ou moins finement les accès aux fichiers. Malheureusement, dans la pratique, les ACL et les permissions étendues ne sont que rarement utilisées, empêchant les utilisateurs de gérer avec précision les autorisations d'accès à leurs propres fichiers. En addition, il est nécessaire de prévoir que certains utilisateurs d'un système (voire même la majorité) ne connaîtra pas les détail des droits d'accès Unix standard, en particulier s'ils ne sont pas informaticiens. Il est donc indispensable que l'administrateur système leur facilite la tâche et les informe le plus possible. La première chose à faire est de choisir une valeur adaptée pour le `umask`, de manière à ce que leurs fichiers soient le plus protégés possible, mais ceci n'est pas suffisant.

Une bonne technique, utilisée par Marc Schaefer sur les serveurs d'`alphanet.ch` consiste à créer d'office deux sous-répertoires, `private` et `public`, dans le répertoire personnel de chaque utilisateur au moment où ce dernier est ajouté au système, et de spécifier des permissions adaptées afin que les utilisateurs puissent facilement classer leurs informations personnelles selon l'accès qu'ils veulent donner aux autres :

```
drwx------    2 user     user         4096 Oct 15 00:01 private/
drwxr-xr-x    2 user     user         4096 Oct 15 00:01 public/
```

Il est important de noter qu'au moment de déterminer si un accès à un fichier est légitime ou non, il ne suffit pas de prendre en compte des considérations techniques, mais aussi, et surtout, d'appliquer des principes de bonne foi, sur le même principe que si la porte d'une maison est ouverte, cela ne signifie pas que vous êtes autorisé à y entrer et à prendre tout ce que vous y trouvez ! Ainsi, même si ses permissions UNIX l'indiquent comme "accessible à tout le monde" (`r--r--r--`), aucun utilisateur ne devrait accéder au fichier `mail` d'un autre utilisateur.

Afin de clarifier les idées de tout le monde, il est fortement conseillé de distribuer aux nouveaux utilisateurs un document résumant quelques informations techniques de base sur les moyens de protéger leurs données personnelles, ainsi que sur ce qui est accepté ou non moralement sur un système.

3 La gestion des mots de passe sous Unix

Pour des raisons de sécurité, les mots de passe des utilisateurs Unix ne sont pas stockés tels quels, mais uniquement sous forme d'une "empreinte" dans le fichier `/etc/passwd`, comme par exemple :

$$\underbrace{\texttt{schutz}}_{\text{username}} : \underbrace{\texttt{MN}}_{\text{salt}}\underbrace{\texttt{ax0hcJfaOHA}}_{\text{empreinte}} : \underbrace{\texttt{1211}}_{\text{uid}} : \underbrace{\texttt{350}}_{\text{gid}} : \underbrace{\texttt{SCHUETZ Frederic}}_{\text{description}} : \underbrace{\texttt{/user/l1/schutz}}_{\text{home dir}} : \underbrace{\texttt{/bin/csh}}_{\text{shell}}$$

La caractéristique principale de cette empreinte est qu'elle est facile à calculer à partir du mot de passe, mais qu'il est quasiment impossible de retrouver le mot de passe à partir de l'empreinte. Au moment où un utilisateur veut s'authentifier, il donne son mot de passe, dont l'empreinte est calculée, puis comparée avec celle stockée pour accorder ou non l'accès.

La seule attaque possible consiste à essayer tous les mots de passe possibles, en commençant par les plus plausibles (par exemple les mots du dictionnaire) et calculer leur empreinte pour voir si elle

correspond à celle disponible dans le fichier `/etc/passwd`. Beaucoup d'utilisateurs choisissent des mots de passe simples à mémoriser, mais également simples à découvrir, ce qui rend cette attaque efficace et aide énormément les pirates.

Nous allons voir quelles sont les méthodes que l'on peut mettre en œuvre pour éviter ces problèmes.

3.1 PAM (Pluggable Authentification Modules)

La plupart des distributions Linux incluent désormais un système d'authentification nommé PAM (Pluggable Authentification Modules) qui vous permet de configurer facilement les méthodes d'authentification utilisées par le système, sans avoir besoin de recompiler des programmes tels que `/bin/login`. Les exemples ci-dessous utilisent la plupart du temps les possibilités de PAM, mais celles-ci sont largement plus grandes : vous pouvez mettre des limites sur les ressources disponibles pour vos utilisateurs (nombres de processus, mémoire disponible, ...), n'autoriser certains d'entre eux à ne se connecter qu'à certains moments depuis certains endroits, voire créer vos propres modules d'authentification si ceux disponibles ne vous suffisent pas.

3.2 Choisir un bon mot de passe

La sécurité d'un mot de passe est difficile à évaluer, et doit faire l'objet d'un compromis entre "pas simple à deviner", et "trop dur à mémoriser", auquel cas l'utilisateur l'écrira sur un post-it, ruinant tous les efforts de sécurité.

Un bon mot de passe doit satisfaire plusieurs conditions :

- être suffisamment long (6 caractères est un miminum)
- être suffisamment complexe (comprendre des lettres majuscules, minuscules, des chiffres et éventuellement d'autres caractères)
- ne correspondre à aucun nom, surnom, numéro, lieu, ou quoi que ce soit qui puisse être lié à vous
- ne correspondre à aucun mot, dans quelque langue que ce soit.
- n'être utilisé que sur un système à la fois.

Le mot de passe idéal est totalement aléatoire, choisi par une machine et pas par un humain, ce qui peut le rendre difficile à mémoriser. La condition "totalement aléatoire" peut néanmoins être difficile à garantir pour une machine, et il arrive qu'un tel générateur ne puisse générer qu'un petit nombre de mots de passe.

Une bonne alternative consiste à mémoriser une phrase et à en tirer un mot de passe en prenant la première lettre de chaque mot. Par exemple, la phrase "un bon chat vaut mieux que deux tuyaux gras", pourrait conduire à un bon mot de passe tel quel "`1bc>2TG`".

Une autre solution originale existe, qui vous permettra en particulier de résister à la tentation d'utiliser plusieurs fois le même mot de passe. Il s'agit d'une petite carte, format carte de crédit, qui contient plusieurs alphabets disposés aléatoirement, et vous permet de choisir des mots de passe de façon aléatoire et de s'en souvenir ensuite. On trouvera plus d'informations sur `http://www.pathword.com`.

Le programme `passwd` peut effectuer des tests et refuser les mots de passe qui seraient trop courts, trop simples ou connus. La configuration se fait dans le fichier `/etc/pam.d/passwd`, avec les options `min` et `max` :

```
password   required   pam_unix.so nullok obscure min=4 max=8
```

3.3 Lancer les attaques soi-mêmes

Une caractéristique importante des outils d'attaques des pirates est qu'ils sont toujours à double tranchant : n'hésitez donc jamais à les utiliser pour rechercher vous-même les mots de passes trop faciles sur votre système et avertir vos utilisateurs qu'ils doivent les changer. Plusieurs de ces outils sont disponibles.

Le premier est `crack` d'Alex Muffett, qui n'existe pas encore sous forme de package Debian, mais que l'on peut trouver à l'adresse `http://www.users.dircon.co.uk/~crypto/`. Une fois que vous avez créé un dictionnaire de mots, il suffit de lancer `crack fichier_passwd` pour qu'il essaie de trouver les mots de passe correspondant. Attention, suivant le nombre de mots de passe à découvrir et la taille du dictionnaire, l'exécution peut prendre des jours, voire des mois !

Il existe également une bibliothèque de fonctions nommée `cracklib`, du même auteur, qui peut être utilisée par la commande `passwd` pour vérifier au moment du choix d'un mot de passe si celui-ci est suffisamment sûr. La configuration se fait dans le fichier `/etc/pam.d/passwd`, avec une ligne telle que :

```
password required       pam_cracklib.so retry=3 minlen=6 difok=3
```

3.4 Shadow passwords

Une protection évidente est de ne pas laisser l'accès aux empreintes elle-même, pour éviter que les pirates ne puissent facilement tester les mots de passe. Il n'est pas possible de simplement interdire aux utilisateurs de lire le fichier `/etc/passwd`, car il contient des informations indispensables (tels que la liaison entre un UID et le nom de l'utilisateur correspondant) pour beaucoup d'applications.

La solution adoptée s'appelle "shadow passwords". Elle consiste à scinder le fichier `/etc/passwd` en deux parties, en supprimant l'empreinte du mot de passe du fichier original, duquel ne restera que les autres informations :

```
schutz:x:1211:350:SCHUETZ Frederic:/user/l1/schutz:/bin/csh
```

et en créant un nouveau fichier `/etc/shadow`, interdit en lecture pour les utilisateurs, qui contiendra les empreintes (avec ici, des informations d'expiration en plus) :

```
schutz:MNaxOhcJfaOHA:11240:0:99999:7:::
```

et qui aura les permissions suivantes :

```
-rw-r-----    1 root     shadow        700 Oct 10 20:12 /etc/shadow
```

Il est évidemment interdit en lecture pour les autres utilisateurs. L'utilisation d'un groupe `shadow` spécifique permet de donner accès en lecture à quelques programmes sûrs (ou en tout cas qu'on espère être sûrs...) qui en auraient besoin, par exemple `xlock`.

La plupart des distributions Linux proposent par défaut lors de l'installation d'installer les mots de passe shadow, ce qui est une bonne chose. Si vous n'avez pas choisi cette option, ou si vous voulez revenir à l'ancienne méthode, les commandes `pwconv` (PassWord CONVert) et `pwunconv` vous permettent de transformer votre fichier `passwd` en fichier `shadow` et vice-versa.

3.5 Utiliser un meilleur système d'empreintes

Le système d'empreintes généralement installé est basé sur DES (Data Encryption Standard), le standard de chiffrement symétrique du gouvernement américain, qui est utilisé de la manière suivante : seuls les 8 premiers caractères du mot de passe sont pris en considération et pour chacun d'eux, on ne garde que les 7 premiers bits (sur 8), soit un total de 56 bits, qui forment une clé de

chiffrement pour DES. On utilise cette clé pour chiffrer une chaîne de caractères standard, le plus souvent "00000000". Le résultat chiffré est l'empreinte du mot de passe. Un paramètre aléatoire, le *salt* est également utilisé lors du chiffrement, et stocké ensuite avec le mot de passe. Avec ses 4096 possibilités, le but est d'empêcher que le même mot de passe choisi par deux utilisateurs différents ait la même empreinte. Ceci complique également le travail d'un pirate, qui ne pourra pas chercher les mots de passe de tous les utilisateurs en même temps, mais sera contraint d'utiliser un salt différent pour chaque utilisateur.

Les nombreuses recherches menées depuis plus de 25 ans sur DES semblent montrer qu'il n'est pas possible de retrouver la clé (le mot de passe) à partir du résultat chiffré (l'empreinte). Si l'algorithme est bon, son implémentation possède néanmoins plusieurs faiblesses. En particulier, la taille effective d'un mot de passe ne dépasse pas celle d'une clé DES, 56 bits, et le nombre total de mots de passe possibles n'est que de $2^{56} \simeq 10^{17}$, ce qui devient à la portée d'une recherche exhaustive.

Un autre système d'empreinte, MD5 (Message Digest 5) est fourni en standard avec les distributions Linux depuis quelques temps. Il a été conçu comme système d'empreinte dès le départ, et n'est donc pas un système de cryptographie qui a été bricolé par la suite. MD5 utilise entièrement le mot de passe, et élimine le risque d'une attaque exhaustive — seule une attaque au dictionnaire doit être prise en compte.

Debian propose à l'installation d'utiliser le système MD5, mais l'option par défaut reste DES, pour des raisons de compatibilité. Si vous changez d'avis après coup, il vous faut modifier le fichier `/etc/pam.d/passwd`, pour que la ligne

```
password   required   pam_unix.so md5 nullok obscure min=4 max=8
```

contienne ou non l'option `md5`.

3.6 Expiration des mots de passe

Le superutilisateur peut spécifier une durée de validité limitée pour le mot de passe d'un utilisateur, à l'aide de la commande `passwd options utilisateur` et avec les options suivantes :

- `-x jours` définit le nombre maximal de jours avant qu'un mot de passe expire et que l'utilisateur soit obligé de le changer.
- `-n jours` définit le nombre minimum de jours avant qu'un mot de passe ne puisse être changé.
- `-w jours` définit le nombre de jours avant l'expiration pendant lesquels un avertissement sera donné à l'utilisateur pour le faire changer de mot de passe.
- `-i jours` définit combien de jours après l'expiration le compte de l'utilisateur sera bloqué.

Les informations d'expiration sont stockées dans le fichier `/etc/passwd` ou `/etc/shadow`.

4 Chroot

Il arrive que l'on doive utiliser ou essayer un programme en lequel on n'a pas entièrement confiance, et qu'on ne désire pas prendre le risque d'abimer une partie de son système. L'utilitaire `chroot` (CHange ROOT), qui n'est utilisable que par `root` et qui s'appelle avec

```
chroot racine programme
```

permet d'isoler ce `programme` dans le répertoire `racine`, duquel il ne pourra pas sortir, étant persuadé d'être dans le répertoire racine du système. Ainsi, tout le reste de l'arboresence du système lui sera inaccessible.

Ceci signifie également que si le programme devait accéder à des fichiers extérieurs à son nouveau répertoire racine, il ne le pourra plus ! Si le programme `exemple`, qui utilise le fichier de configuration `/etc/exemple.conf` est lancé en `chroot` dans le répertoire `/chroot/exemple/`, vous serez obligé

de recréer une mini-hiérarchie avec le fichier `/chroot/exemple/etc/exemple.conf`, pour que le programme puisse accéder à ce qu'il pense être `/etc/exemple`. Ceci est particulièrement vrai pour les bibliothèques dynamiques stockées dans `/lib` : la plupart des programmes ne seront plus capables de fonctionner sans y avoir accès ! (pour voir de quelles bibliothèques a besoin un programme, utiliser la commande `ldd -v programme`)

L'utilisation la plus importante de `chroot` est pour des programmes tels que `bind` ou `ftpd` (pour des accès ftp anonymes), qui sont coutumiers des trous de sécurité, et pour lesquels on désire diminuer l'impact d'un nouveau problème. Pour `bind`, le "Chroot-BIND HOWTO" (disponible sur `http://www.linuxdoc.org`) explique en détail la manière de procéder. Le serveur `ftpd` possède une option `-r racine` pour se restreindre automatiquement à un répertoire donné.

Il ne faut pas perdre de vue que la sécurité de `chroot` est illusoire si l'utilisateur est `root`. Par exemple, si des fichiers spéciaux tels que `/dev/kmem` ou `/dev/sda3` sont accessibles, le processus peut écrire directement dans la mémoire ou sur le disque, sans passer par le système de fichiers. Dans le premier cas, il pourrait changer à nouveau son répertoire `chroot`, et sortir de sa prison. Dans le deuxième cas, il pourrait changer des fichiers (par exemple un mot de passe) ou leurs permissions, en prévision d'une attaque future. Il faut donc éviter de faire tourner des processus avec l'identité de `root` en `chroot`. Si les privilèges `root` sont nécessaires pour accéder à un port privilégié, on peut utiliser à la place les possibilités de redirection de port du noyau (voir `ipfwadm` dans le noyau 2.2).

Selon la configuration, un programme qui n'a pas les privilèges `root` pourra aussi sortir de sa cage (voire se connecter à une autre machine) en utilisant les équivalences réseaux `rhosts` ou `ssh` : un simple `rlogin localhost` peut alors lui permettre d'accéder à toute l'arborescence. Ces services sont donc à configurer avec prudence, voire à éliminer dans le cas des `rhosts`.

Une autre utilisation du système `chroot` consiste à créer un serveur de test à l'intérieur de votre système afin d'essayer des programmes inconnus. On peut par exemple décompresser le package `base2_2.tgz` de Debian dans un sous-répertoire, puis lancer un shell limité à ce sous-répertoire avec

```
chroot /chroot/essai su user -c "bash --login"
```

pour pouvoir ensuite essayer tout ce qu'on veut sans risque pour le système

On peut également utiliser `chroot` si on est obligé de démarrer avec une disquette de secours suite à un problème. On monte ensuite la partition racine habituelle sur `/mnt`, puis on fait un `chroot` afin de se retrouver dans les conditions normales, sans avoir besoin de changer les chemins d'accès des fichiers de configuration — par exemple si on doit réinstaller `lilo`. Dans ce dernier cas, `lilo` peut aussi effectuer directement un `chroot` au démarrage en utilisant l'option `-r racine`.

5 Quelques problèmes pratiques

1. Faire une copie de l'utilitaire `id` (habituellement dans `/usr/bin/id`), et lui donner la permission SUID `root`. Regarder ensuite en tant qu'utilisateur la différence à l'exécution entre l'ancienne commande et la nouvelle. Essayer de faire la même chose sur une partition qui a été montée avec l'option `nosuid`.
2. Pour améliorer la sécurité d'un système, on n'autorise pas en général tous les utilisateurs à utiliser la commande `su` : seuls les membres du groupe `wheel` sont autorisés à utiliser cette commande. Comment peut-on mettre en place un tel mécanisme ?
3. Dans le même genre d'idées, certains utilisateurs ont besoin d'accéder à des utilitaires SUID root (par exemple `ping`), sans avoir la possibilité d'être root eux-même, et sans donner cet accès à tous les utilisateurs. Comment peut-on mettre en place un tel système ?
4. Que se passe-t-il si un shell, `/bin/sh` par exemple, possède le bit SUID root ? Comment peut-on permettre à certaines personnes d'accéder au compte root en tapant une simple commande,

sans mot de passe ? Expliquer pourquoi c'est une mauvaise idée. Essayer également avec `/bin/bash`.

5. Un utilisateur a créé quelques pages web dans son répertoire `~user/www`, et le serveur web a été configuré pour accéder et publier ces pages, mais seulement après introduction d'un mot de passe.

   Seulement, il se rend compte que les autres utilisateurs de son système n'ont qu'à faire un `cd ~user/www` pour accéder aux pages. Mais il ne peut pas simplement enlever leurs droits de lecture, car le serveur web, qui tourne sous son propre utilisateur et groupe, ne pourrait plus y accéder lui-même. Comment faire pour empêcher l'accès aux utilisateurs, tout en le laissant au serveur et en comptant que chaque utilisateur désire faire pareil ?

   Indications. Il y a (au moins) deux solutions. La première nécessite que le serveur web et l'utilisateur fassent partie d'un même groupe, et que personne d'autre n'en fassent partie.

   La deuxième est de donner au répertoire `~user/www` les permissions de groupes nécessaires pour que le serveur web puisse le lire, que l'utilisateur puisse y écrire, et que tous les nouveaux fichiers de ce répertoire héritent de ces permissions.

6. Vérifiez si votre mot de passe est un "bon" mot de passe. Même s'il satisfait les conditions, cherchez-en un plus compliqué.

6 Références

- Fenzi Kevin et Dave Wreski, *Security-HowTo*. Disponible sur `http://www.linuxdoc.org`.
- Garfinkel Simson et Gene Spafford, *Practical Unix Security* 2nd edition. O'Reilly, avril 1996. ISBN 1-56592-124-0
- `http://www.linuxsecurity.com`, en particulier `http://www.linuxsecurity.com/docs`

Correction des travaux pratiques

1. Faire une copie de l'utilitaire `id` (habituellement dans `/usr/bin/id`), et lui donner la permission SUID `root`. Regarder ensuite en tant qu'utilisateur la différence à l'exécution entre l'ancienne commande et la nouvelle. Essayer de faire la même chose sur une partition qui a été montée avec l'option `nosuid`.

   Solution. On voit que la présence de la permission SUID change effectivement l'EUID du processus, sans changer son UID ni ses groupes. On remarque aussi que le programme ne peut pas être lancé sur la partition `nosuid`.

```
schutz@laptop:/tmp$ su
Password:
laptop:/tmp# cp /usr/bin/id .
laptop:/tmp# chmod u+s id
laptop:/tmp# mount
[...]
/dev/hda7 on /home type ext2 (rw,nosuid)
laptop:/tmp# cp ./id /home/schutz
laptop:/tmp# exit
exit
schutz@laptop:/tmp$ ls -l id
-rwsr-xr-x    1 root     root         9552 Nov 27 15:52 id*
schutz@laptop:/tmp$ id
uid=1000(schutz) gid=1000(schutz) groups=1000(schutz),40(src)
schutz@laptop:/tmp$ ./id
uid=1000(schutz) gid=1000(schutz) euid=0(root) groups=1000(schutz),40(src)
schutz@laptop:/tmp$ ls -l /home/schutz/id
-rwsr-xr-x    1 root     root         9552 Nov 27 15:56 /home/schutz/id*
schutz@laptop:/tmp$ /home/schutz/id
bash: /home/schutz/id: Operation not permitted
```

2. Pour améliorer la sécurité d'un système, on n'autorise pas en général tous les utilisateurs à utiliser la commande `su` : seuls les membres du groupe `wheel` sont autorisés à utiliser cette commande. Comment peut-on mettre en place un tel mécanisme ?

   Solution. On commence par créer un nouveau groupe appelé `wheel` et on met dedans les utilisateurs que l'on veut :

```
schutz@laptop:/etc$ su
Password:
laptop:/etc# addgroup wheel
laptop:/etc# adduser wheel schutz
```

   Attention, l'utilisateur `schutz` doit se déloguer entièrement puis se reloguer pour que son nouveau groupe soit pris en compte.

   On attribue ensuite la commande `su` au groupe `wheel` et on interdit son exécution par d'autres utilisateurs :

```
laptop:/etc# ls -l /bin/su
-rwsr-xr-x    1 root     root        23420 Oct 10 21:33 /bin/su
laptop:/etc# chgrp wheel /bin/su
```

```
laptop:/etc# chmod 4750 /bin/su
laptop:/etc# ls -l /bin/su
-rwsr-x---    1 root     wheel       23420 Oct 10 21:33 /bin/su
```

Attention, le fait de changer le groupe de `su` lui fait perdre son bit SUID qu'il est indispensable de recréer !

Depuis là, l'utilisateur `schutz` peut l'utiliser :

```
schutz@laptop:~$ su
Password:
laptop:/home/schutz#
```

mais il est inaccessible pour les autres utilisateurs :

```
user@laptop:~$ su
sh: /bin/su: Permission denied
```

Il est également possible d'utiliser le module `pam_wheel` de PAM pour qu'il n'autorise l'accès à `su` qu'aux membres du groupe `wheel`. Cela se fait en rajoutant la ligne

```
auth       required   pam_wheel.so
```

au fichier `/etc/pam.d/su`. Il est également possible de configurer ce module plus finement, pour interdire l'accès à `su` à certains groupes ou dispenser de mot de passe les utilisateurs du groupe `wheel`, ce qui n'est pas recommandé.

3. Dans le même genre d'idées, certains utilisateurs ont besoin d'accéder à des utilitaires SUID root (par exemple `ping`), sans avoir la possibilité d'être root eux-même, et sans donner cet accès à tous les utilisateurs. Comment peut-on mettre en place un tel système ?

   Solution. C'est exactement le même principe qu'à l'exercice précédent, sauf qu'il faut inventer un groupe pour l'occasion — `pingok` peut être un bon choix.

4. Que se passe-t-il si un shell, `/bin/sh` par exemple, possède le bit SUID root ? Comment peut-on permettre à certaines personnes d'accéder au compte root en tapant une simple commande, sans mot de passe ? Expliquer pourquoi c'est une mauvaise idée. Essayer également avec `/bin/bash`.

   Solution. On voit que si `sh` possède le bit SUID, alors n'importe quel utilisateur qui l'exécute devient `root` sans avoir besoin de mot de passe :

   ```
   laptop:/tmp# cp /bin/sh .
   laptop:/tmp# chmod u+s sh
   laptop:/tmp# exit
   exit
   schutz@laptop:/tmp$ ./sh
   schutz@laptop:/tmp# id
   uid=1000(schutz) gid=1000(schutz) euid=0(root) groups=1000(schutz),40(src)
   ```

   Si on utilise le même principe que dans les exercices précédents, on peut obtenir l'équivalent d'une commande `su` dont l'accès est limité aux utilisateurs du groupe `wheel` et qui ne nécessite pas de mot de passe :

   ```
   laptop:/tmp# chgrp wheel sh
   laptop:/tmp# chmod 4750 sh
   laptop:/tmp# ls -l sh
   -rwsr-x---    1 root     wheel      461400 Nov 27 16:32 sh
   ```

   C'est une mauvaise idée, car ça revient à dire qu'il suffit d'arriver à pirater le compte d'un utilisateur du groupe `wheel` pour automatiquement pirater le compte de `root`. On augmente donc les portes d'entrée vers `root`, ce qui est une mauvaise chose.

   Si on essaie avec `bash`, on remarque que ça ne marche pas, car il détecte que UID≠EUID et abandonne par sécurité son identité de `root` au démarrage.

```
laptop:/tmp# cp /bin/bash .
laptop:/tmp# chmod u+s bash
laptop:/tmp# exit
exit
schutz@laptop:/tmp$ ./bash
schutz@laptop:/tmp$ id
uid=1000(schutz) gid=1000(schutz) groups=1000(schutz),40(src)
```

5. Un utilisateur a créé quelques pages web dans son répertoire `~user/www`, et le serveur web a été configuré pour accéder et publier ces pages, mais seulement après introduction d'un mot de passe.

   Seulement, il se rend compte qu'il suffit aux autres utilisateurs de faire un `cd ~user/www` pour accéder aux pages. Mais il ne peut pas simplement enlever leurs droits de lecture, car le serveur web, qui tourne sous son propre utilisateur et groupe, ne pourrait plus y accéder non plus. Comment faire pour empêcher l'accès aux utilisateurs, tout en le laissant au serveur et en comptant que chaque utilisateur désire faire pareil ?

   Indications. Il y a (au moins) deux solutions. La première nécessite que le serveur web et l'utilisateur fassent partie d'un même groupe, et que personne d'autre n'en fassent partie.

   La deuxième est de donner au répertoire `~user/www` les permissions de groupes nécessaires pour que le serveur web puisse le lire, que l'utilisateur puisse y écrire, et que tous les nouveaux fichiers de ce répertoire héritent de ces permissions.

   Solution. La première solution utilise le fait que dans certaines distributions, chaque utilisateur est dans son propre groupe. L'administrateur système ajoute ensuite le serveur web (qui s'exécute ici sous l'identité de `www-data`) dans le groupe de l'utilisateur à l'aide de la commande `adduser www-data schutz`. Il suffit ensuite que l'utilisateur protège son répertoire `~user/www` de façon restrictive pour qu'aucun autre utilisateur ne puisse y accéder :

   ```
   drwxr-x---    2 schutz   schutz       4096 Nov 27 16:42 www/
   ```

   La protection des fichiers à l'intérieur de ce répertoire est peu importante (tant qu'ils sont lisibles par le groupe), vu qu'un autre utilisateur ne peut même pas rentrer dans le répertoire.

   La deuxième solution utilise le groupe du serveur web, qui est aussi `www-data` normalement. L'administrateur système change les permissions du répertoire `~user/www` de manière à ce qu'il appartienne au groupe `www-data`, lui ajoute la permission SGID pour que les fichiers du répertoire y soient aussi, et interdit l'accès aux autres utilisateurs :

   ```
   schutz@laptop:~$ ls -l
   [...]
   drwxr-x---    2 schutz   schutz       4096 Nov 27 16:42 www/
   schutz@laptop:~$ su
   Password:
   laptop:/home/schutz# chgrp www-data www
   laptop:/home/schutz# chmod 2750 www
   laptop:/home/schutz# exit
   exit
   schutz@laptop:~$ ls -l
   [...]
   drwxr-s---    2 schutz   www-data     4096 Nov 27 16:42 www/
   ```

   L'utilisateur ne reçoit aucun droit supplémentaire (il ne peut toujours pas lire les autres fichiers qui appartiennent au groupe `www-data`), mais le serveur web peut maintenant lire ses fichiers, sans que les autres utilisateurs ne le puissent.

   On peut voir que le bit SGID produit bien l'effet désiré à l'intérieur du répertoire, et qu'il s'hérite même sur les sous-répertoires :

   ```
   schutz@laptop:~$ touch essai
   ```

```
schutz@laptop:~$ ls -l essai
-rw-r--r--    1 schutz   schutz          0 Nov 27 16:51 essai
schutz@laptop:~$ touch www/essai
schutz@laptop:~$ mkdir www/sous-www
schutz@laptop:~$ ls -l www
total 4
-rw-r--r--    1 schutz   www-data        0 Nov 27 16:51 essai
drwxr-sr-x    2 schutz   www-data     4096 Nov 27 16:52 sous-www/
```

Bien entendu, la sécurité de ces systèmes dépend aussi de la configuration du serveur web `apache`. Par exemple, l'utilisation de l'option `Option SymLinksIfOwnerMatch`, qui évite qu'un utilisateur ait le droit de faire un lien symbolique sur les fichiers d'un autre utilisateur, est indispensable. Sinon, avec une commande telle que :

```
user@laptop:~/www$ ln -s /home/schutz/www lien
```

un utilisateur créerait une page `http://localhost/~user/lien` qui pointe en fait sur le répertoire qui est censé être protégé, mais qu'Apache peut lire et transmettra.

Dans ce cas comme toujours, la sécurité globale n'est pas plus forte que le maillon le plus faible de la chaîne !