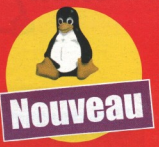
LINUX SCHOOL
Magazine



N° 2 / FEVRIER-MARS 2008 / 4.50 EUROS

HACKING LINUX 2

Sécurisation

Crypter fichiers et e-mails

Heavy Firewalling

Sommaire

LINUXSCHOOL MAGAZINE est édité par LPN 15 RUE CHEVREUL - 94 700 MAISONS-ALFORT
Rédaction en chef : Linux Community • **Directeur de Publication et représentant légal :** André Olivier
Imprimé en France par ROTO GARONNE 47310 Estillac • La Rédaction accepte toutes les contributions de la Communauté
Commission paritaire en cours • Dépôt légal à parution • ISSN en cours •

Le système de fichiers selon Linux

Un système linux, c'est un noyau qui prend en charge le matériel, en contact avec un espace utilisateur, le shell. Le shell a pour mission de recevoir les commandes des utilisateurs et de lancer les applications. Mais linux, c'est aussi un système de fichiers hiérarchisé, pratiquement toujours le même, quel que soit l'UNIX utilisé, à quelques variantes près. La distribution Suse(7) par exemple utilise une arborescence tellement bordélique que je n'ose même plus accepter d'administrer des machines sous cette distribution. Cela dit, la Suse a quand même un certain nombre d'avantages. Dans un environnement UNIX, tout est fichier. C'est une notion relativement difficile à comprendre et à admettre : les fichiers sur votre disque dur, votre écran, votre carte son, votre imprimante, tout cela est fichier. Le noyau communique avec votre matériel via un fichier présent sur votre disque dur nommé un descripteur de fichier. C'est un peu plus compliqué que cela dans la réalité, mais si, déjà, vous arrivez à admettre que votre écran soit un fichier, et non plus du matériel, c'est déjà pas mal. Une fois qu'on a pris le pli, cette caractéristique d'UNIX devient un atout indispensable dès qu'on veut s'attaquer à la programmation système. Le noyau communique avec le hardware via des points de montage. Il s'agit tout simplement de fichiers auxquels l'utilisateur a la possibilité d'accéder, et qui pointent vers ce matériel, une fois monté. Le montage est l'action de dire au noyau "il existe sur ma machine tel matériel auquel tu peux accéder via tel descripteur, et que je veux accéder via tel fichier". Le noyau va alors mettre le matériel concerné à disposition de l'utilisateur là où il le lui a demandé. Le disque dur lui-même est un fichier, monté sur un point de montage spécifique. Le disque principal, ou racine, sur lequel tout le reste va venir se greffer (des fichiers, bien évidemment) a pour point de montage le répertoire racine, ou "/".

L'arborescence des répertoires

L'arborescence des répertoires est pratiquement toujours la même :

```
/
  --> /bin
  --> /boot
  --> /dev
  --> /etc
  --> /home
  --> /lib
  --> /mnt
  --> /proc
  --> /root
  --> /sbin
  --> /tmp
  --> /usr
        --> /X11R6
        --> /bin
        --> /include
        --> /lib
        --> /local
        --> /man
        --> /sbin
        --> /share
        --> /src
```

```
--> /var
        --> /log
        --> /spool
--> swap
```

Cette arborescence est loin d'être exclusive, et de nombreux autres répertoires peuvent exister. Mais il s'agit de l'arborescence de base sur une distribution linux. Tous les exemples proposés dans ce cours ont été effectués sur une slackware 8.0. Nous allons maintenant voir à quoi correspondent ces différents répertoires.

- / : c'est le répertoire racine (rien à voir avec Jayce et ses monstres aux plantes). C'est le premier répertoire monté au démarrage, et celui sur lequel tous les autres vont être montés.

- /bin : c'est le répertoire où le système installe la grande majorité des programes utilisables par les utilisateurs normaux.

- /boot : ce répertoire est capital car c'est là que les éléments nécessaires au démarrage seront entreposés, notamment le noyau, ou un lien pointant vers lui.

- /dev : c'est là que se trouvent les entrées vers les différents périphériques, qui permettent au noyau d'accéder au matériel. Il s'agit là des points d'accès et non pas des points de montage.

- /etc : c'est là que se trouvent la plupart des fichiers de configuration du système, ou encore les scripts lancés au démarrage.

- /home : c'est là que sont stockés les répertoires des utilisateurs (ou home directories).

- /lib : c'est là que sont installées la plupart des bibliothèques nécessaires au bon fonctionnement du système.

- /mnt : c'est là que se trouvent, en général, les points de montage vers les disques physiques, comme les disques durs amovibles, ou encore les lecteurs de cdrom une fois montés. En effet, pour être accessibles, les cdroms doivent d'abord être montés, et ne peuvent pas être retirés s'ils n'ont pas été démontés d'abord. Normalement, seul le superutilisateur peut monter du matériel, mais de nombreux scripts autorisent l'utilisateur à le faire automatiquement sur les distributions grand public.

- /proc : c'est là que vous trouverez les informations sur le fonctionnement de vos périphériques. C'est extrêmement barbare à lire, mais cela peut s'avérer utile en cas de dépannage, ou de simple curiosité.

- /root : c'est le répertoire personnel du superutilisateur. Il ne se trouve pas dans /home, mais à la racine du disque.

- /sbin : c'est le répertoire dans lequel le système a installé les programmes accessibles uniquement au superutilisateur.

- /tmp : c'est là que ce trouvent les fichiers temporaires créés par le système.

- /usr : c'est là que se trouvent tous les programmes non systèmes et librairies accessibles aux utilisateurs. Il contient de très nombreux sous-répertoires, mais les plus importants sont :
  --> /usr/X11R6 : c'est là que se trouvent les binaires du système d'affichage X windows, l'interface graphique la plus communément disponible sous linux.
  --> /usr/bin : c'est là que se trouvent les exécutables (ou binaires) des programmes utilisateurs.
  --> /usr/include : c'est là que se trouvent les entêtes des librairies installées dans l'espace utilisateur. Ces entêtes sont nécessaires dès qu'il est question de recompiler des programmes.
  --> /usr/lib : c'est là que se trouvent les librairies utilisées par les applications des utilisateurs.
  --> /usr/local : c'est là que sont les programmes installés par les utilisateurs de la machine. Tout comme /usr, /usr/local dispose d'un répertoire bin, include, lib, etc. etc. etc.

--> /usr/man : c'est là que se trouvent les anpages, une source de documentation intarissable sur les programmes présents sur le système ou encore sur les différentes bibliothèques. Avant de poser une question, le bon réflexe est de se jeter sur les mans, ou encore sur les howto de la Linux Documentation Project(8), qui décrivent bien des choses pas à pas.
--> /usr/sbin : c'est là que se trouvent les programmes non système réservés au super utilisateur, et qui font appel à des ressources auxquelles les utilisateurs normaux n'ont pas accès.
--> /usr/share : c'est là que se trouvent les programmes partagés par les utilisateurs, selon l'arborescence traditionnelle d'UNIX. Mais ces règles sont de moins en moins respectées.
--> /usr/src : ce répertoire est le préféré de Monsieur Bidouilleur car c'est là que le root entrepose les sources des programmes qu'il va installer sur son système.

- /var : c'est dans ce répertoire que vont se retrouver tous les répertoires de spoule, comme celui de l'imprimante ou du serveur de mail. Un répertoire de spoule est un répertoire dans lequel un programme ou un périphérique va mettre en attente les informations qui devront être traitées plus tard.
--> /var/spool : c'est là que se trouvent les répertoires de spoule. Les deux plus importants sont /var/spool/lpd, où les travaux d'impression sont mis en attente, et /var/spool/mail où les mails sont mis en attente de lecture.
--> /var/log : c'est là que sont entreposés les fichiers log du système. Mais ces fichiers ne sont pas accessibles aux utilisateurs.

- la swap : il s'agit de la mémoire tampon, qui utilise une partition pour elle seule et ne peut pas être lue ou écrite directement par un être humain. Elle vient suppléer la mémoire vive de l'ordinateur car linux, s'il ne demande pas un processeur rapide, demande beaucoup de mémoire.

"La différence entre connaître le chemin et arpenter le chemin"

Nous allons nous pencher ici sur la notion de chemin, ou path en bon anglais. En effet, si vous vous penchez sur votre fichier .profile, situé soit dans votre répertoire racine soit dans le répertoire /etc, vous verrez une ligne commençant par PATH=

```
PATH="/usr/local/bin:/usr/bin:/bin:/usr/X11R6/bin:/usr/sbin:/sbin"
```

Il s'agit là du chemin pour accéder à la plupart des programmes disponibles sur le système sans avoir à taper leur chemin complet. En effet, il est plus simple de lancer "nmap" que "/usr/local/bin/nmap". C'est là que nous allons introduire la différence entre chemin absolu et chemin relatif.

Le chemin absolu est le chemin pris à partir de la racine, tandis que le chemin relatif est celui pris à partir du répertoire courant, c'est-à-dire le répertoire dans lequel on se trouve au moment où on lance la commande.

```
bash-2.05$ /usr/local/bin/nmap
bash-2.05$ ../usr/local/bin/nmap
bash-2.05$ nmap
```

La première ligne lance l'utilitaire nmap en donnant comme accès le chemin depuis la racine, tandis que la seconde ligne indique au shell le chemin depuis le répertoire dans lequel on se trouve. La troisième lance directement nmap car j'ai ajouté le chemin au path pour ne pas perdre de temps à accéder à un utilitaire aussi utile.

Il existe une troisième manière de définir le path, c'est à partir du répertoire courant. Cela se fait en commençant la ligne par "./". Si je me trouve dans le répertoire /usr/local/bin, il me suffira de faire :

```
bash-2.05$ ./nmap
```

La notion d'utilisateur

Comme nous l'avons dit plus haut, linux est un système véritablement multiutilisateur. Cela implique que ce qui appartient à un utilisateur n'appartient pas à un autre, et que n'importe qui ne peut pas faire n'importe quoi n'importe comment, et ça, c'est le pied. Mais il a bien fallu organiser tout ça d'une manière ou d'une autre et c'est ce que nous allons voir dans ce paragraphe.

Je ne suis pas un numéro, je suis un homme libre !!!

Si vous regardez le contenu du fichier /etc/group, vous allez voir une suite de lignes un peu barbares ressemblant à peu près à ça :

```
bash-2.05$ cat /etc/group
root::0:root
bin::1:root,bin,daemon
daemon::2:root,bin,daemon
sys::3:root,bin,adm
adm::4:root,adm,daemon
tty::5:
disk::6:root,adm
lp::7:lp
mem::8:
wheel::10:root
man::15:
mysql::27:
ftp::50:
nobody::98:nobody
nogroup::99:
users::100:
bash-2.05$
```

Nous allons maintenant décrypter le sens de ces lignes un peu étranges. Il s'agit en fait des groupes d'utilisateurs présents sur le système. Chaque utilisateur correspond à un groupe particulier, ce qui lui confère des droits particuliers. Chaque groupe porte un nom et un numéro identifiant pour le système. Ce numéro est le GID ou Group IDentification. A chaque GID correspond un seul groupe et vice-versa. Ainsi, ici, les utilisateurs devraient normalement faire partie du groupe users et avoir le GID 100. Mais bien souvent, par ignorance, les administrateurs créent un groupe par utilisateur, répondant ainsi aux options par défaut du système. Le GID 0 est réservé au super utilisateur qui sera présenté un peu plus loin.

Regardons maintenant le fichier /etc/passwd. Il est encore plus barbare. Il va pourtant falloir le comprendre pour assimiler la notion d'utilisateur sous linux (et sous les différents UNIX).

```
bash-2.05$ cat /etc/passwd
root:x:0:0::/root:/bin/tcsh
bin:x:1:1:bin:/bin:
daemon:x:2:2:daemon:/sbin:
adm:x:3:4:adm:/var/log:
lp:x:4:7:lp:/var/spool/lpd:
sync:x:5:0:sync:/sbin:/bin/sync
ftp:x:14:50::/home/ftp:
mysql:x:27:27:MySQL:/var/lib/mysql:/bin/bash
nobody:x:99:99:nobody:/:
toto42:x:100:100:Toto il est bo:/home/toto42/:/bin/tcsh
bash-2.05$
```

Prenons par exemple la dernière ligne du fichier, qui est celle de l'utilisateur nommé toto42.

- toto42 : il s'agit du login de l'utilisateur, son nom sur la machine. Il peut faire jusqu'à 8 caractères. C'est ce nom que l'utilisateur devra donner quand il voudra avoir accès à la machine.
- x : normalement, à cette place, se trouve le mot de passe crypté de l'utilisateur. Le x signifie que les "shadow passwords" sont installés sur le système. En effet, le fichier /etc/passwd est lisible par n'importe qui. Le système de shadow passwords va déplacer les mots de passe dans un fichier lisible uniquement par le root. Il s'agit de /etc/shadow.
- 100 : il s'agit de l'user ID, ou UID de l'utilisateur. C'est, en quelque sorte, son numéro perso qui permet à la machine de l'identifier. Chaque utilisateur a un seul UID, et le même UID est donné à un seul utilisateur, sauf dans un cas bien précis que nous n'avons pas besoin d'étudier ici (et qui est tellement peu secure que rien que d'y penser, j'en ai les cheveux qui se dressent sur la tête).
- 100 : le second 100 de la ligne correspond au GID de notre utilisateur toto42. Si nous regardons à nouveau /etc/group, nous constatons que toto42 est du group users, ce qui n'a rien de surprenant en soi.
- Toto il est bo : il s'agit du nom réel de l'utilisateur. Celui-ci a choisi un nom parfaitement idiot que l'administrateur lui a laissé. C'est valable sur une machine ayant peu d'utilisateurs, mais au bout d'un certain nombre d'utilisateurs, il vaut mieux rationnaliser un peu tout ça.
- /home/toto42 : il s'agit du répertoire personnel de notre ami toto42. Ce sera son répertoire courant lorsqu'il se loggera.
- /bin/tcsh : c'est le shell, c'est-à-dire l'interpréteur de commandes, choisi par notre ami toto42. Nous verrons cette notion de shell un peu plus loin.

Ze big boss : le root

Le root est le super utilisateur de la machine. Il n'en existe qu'un seul, et il a un certain nombre de caractéristiques bien à lui :

- Un UID bizzare : le root à l'UID 0. Pour le système, cela implique qu'il a tous les droits : lancer des programmes interdits aux autres utilisateurs, ajouter ou retirer un utilisateur, effacer tout et n'importe quoi, y compris la totalité du système.
- Un GID bizzare : il est lui aussi à 0. Normalement, aucun autre utilisateur ne doit avoir ce GID la (pas plus que l'UID 0). Ou alors, soit le root est très négligeant, soit il y a un gros problème et une réinsfallation du système va s'avérer urgente.
- Un homedir à part : il s'agit en effet de /root. En effet, souvent le répertoire utilisateur /home se trouve sur une partition à part. C'est à la fois plus sûr et plus pratique en cas de réinstallation ou de changement de système.
- Tous les droits : nous l'avons dit plus haut, le root a tous les droits. Aussi, il est dangereux de se logger en root, et il vaut mieux ne le faire que pour des opérations ponctuelles. Le mieux est de ne jamais se logger directement et utiliser la commande shell su.

Notions de permissions

Pour gérer les différents utilisateurs et groupes, UNIX a inventé un truc génial : les permissions. Ouvrez une console et listez les fichiers se trouvant dans votre répertoire à l'aide de la commande ls -l.

```
bash-2.05$ ls -l
drwx------    3 toto42   users          72 Oct  6 21:27 mnt
-rwSr--r--    1 toto42   users          11 Sep 23 23:11 pass
drwxr-xr-x    2 toto42   users         184 Sep 22 22:10 pics
lrwxr-xr-x    1 toto42   users         210 Oct  1 11:42 nmap /usr/local/bin/nmap/
bash-2.05$
```

Ce listing est très incomplet, mais parfait pour ce que nous voulons voir. Chacune des lignes est composée de plusieurs colonnes ayant toutes la même utilité. Mais seules les quatre premières nous intéressent ici. Les autres seront expliquées un peu plus loin, lors du détail de la commande ls.

La première colonne est celle qui nous intéresse le plus : il s'agit des permissions du fichier. La ligne se compose de 10 caractères. Le premier se lit seul et nous informe de la nature du fichier :

"-" signifie que le fichier est un fichier des plus normaux.

"l" signifie que le fichier est un lien sur un fichier se trouvant dans un autre répertoire, mais pouvant être accédé directement depuis le répertoire courant. C'est bien pratique pour se rendre plus vite dans un répertoire : un simple lien vers celui-ci suffit.

"d" signifie que le fichier est en fait un répertoire.

Il existe un quatrième type de fichier, qui n'est pas representé ici : le fichier caché. Son nom commence tout simplement par un ".".

La seconde série de caractères va, cette fois, se lire trois par trois : il s'agit des permissions en lecture, écriture et exécution pour le propriétaire du fichier, les membres de son groupe et pour le reste du monde.

- "r" indique que l'utilisateur peut lire le fichier, ou le répertoire.
- "w" indique que l'utilisateur a des droits en écriture sur le fichier. Normalement, seul le propriétaire du fichier devrait pouvoir écrire dessus.
- "x" signifie que le fichier est exécutable. Un répertoire est toujours exécutable, sinon, on ne peut entrer dedans. On ne peut même pas le traverser.

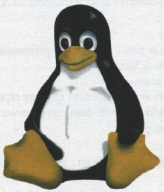
Il existe une quatrième option, très dangereuse mais malheureusement indispensable : "S". Elle signifie que le programme s'exécute non plus au nom de celui qui le lance, mais au nom de celui à qui il appartient. De tels programmes sont dits "setuseridbites" (on prononce ça "setyouseuraildbites". Cela signifie qu'ils ont le bit Suid activé. Si le programme appartient au root, cela peut s'avérer très dangereux, or c'est parfois nécessaire. Et ce sont ces programmes "suid root" que les pirates vont chercher en premier lorsqu'ils voudront pirater votre système.

"Sésame, ouvre-toi"

Sous linux, les mots de passe sont cryptés selon un algorythme non réversible. Cela signifie que, une fois cryptés, les mots de passe ne peuvent plus être décryptés. Lorsque l'utilisateur s'identifie, le système va crypter le mot de passe qu'il a rentré au clavier et va le comparer avec le mot de passe crypté qu'il a dans le fichier /etc/passwd, ou /etc/shadow si les shadow passwords sont installés. S'ils correspondent, tant mieux, sinon, tant pis. Traditionnellement, les mots de passe sous linux font 8 caractères. Mais le système des mots de passe md5 permet maintenant de faire des phrases de passe allant jusqu'a 255 caractères. Un bon truc pour choisir son passe consiste à prendre des majuscules, des minuscules, des chiffres, et des caractères spéciaux. Un exemple parmi d'autres serait "Toto42kicks42ass424242!!!". Prenez de préférence un mot de passe différent pour chaque activité en nécessitant un. Ça peut parfois poser des problèmes de mémoire (ne jamais rien écrire sur un papier ni même dans un fichier informatique), mais ça évite à toutes vos activités électroniques d'être compromises si quelqu'un découvrait votre mot de passe. Et changez-en régulièrement.

Les processus

Lorsqu'un utilisateur lance un programme, il crée un processus qui possède un numéro, un nom et un propriétaire (entre autres). Le numéro du processus se nomme PID. Le processus a aussi un état : actif, endormi et zombi. - actif signifie que le processus est actuellement en activité.
- passif signifie que le processus est au repos, et en attente de réveil.
- zombi signifie que le processus est mort mais que le processus qu'il a lancé (ou processus père) ne le sait pas, et ne peut donc pas l'enterrer. En effet, tout processus est le fils d'un autre, sauf le processus init, qui a pour numéro 1.



Une approche (vraiment très) basique du shell

Si on peut utiliser la plupart des OS en trois minutes sans rien perdre juste en quelques clics, il faut de nombreux logiciels souvent lourds pour être capables de faire ce que peut faire le shell linux. Aussi, utiliser un UNIX sans connaître le shell, c'est un peu comme attacher les lacets d'un joueur de foot : il perd 99% de ses capacités et c'est un beau gâchis. Aussi, nous allons voir ici les 10 commandes indispensables pour utiliser le shell. S'il existe plusieurs shells ayant chacun leurs particularités, les commandes décrites ici sont génériques.

La casse, c'est important

En effet, linux différencie les majuscules et les minuscules, que ce soit dans les commandes, les noms de fichier, les options passées en paramètres des commandes, ou les variables en cas de besoin.

```
bash-2.05$ ls -l
total 0
-rw-r--r--    1 toto42    users       0 Nov 25 22:11 TOTO
-rw-r--r--    1 toto42    users       0 Nov 25 22:11 TOtO
-rw-r--r--    1 toto42    users       0 Nov 25 22:12 TOto
-rw-r--r--    1 toto42    users       0 Nov 25 22:11 ToTO
-rw-r--r--    1 toto42    users       0 Nov 25 22:11 TotO
-rw-r--r--    1 toto42    users       0 Nov 25 22:11 toto
bash-2.05$
```

Tous ces fichiers se nomment toto, mais ils sont tous différents (et vides par ailleurs). Les espaces sont aussi très importants, car un espace en trop peut se révéler absolument catastrophique :

```
bash-2.05$ rm -rf *~
bash-2.05$ rm -rf * ~
```

Ces deux lignes sont à peine différentes et pourtant... La première efface tous les fichiers se finissant par ~ (sauvegardes automatiques de l'éditeur emacs) tandis que la seconde efface l'intégralité de votre home directory.

De même, un simple point oublié peut avoir des conséquences absolument catastrophiques (c'est là qu'on voit à quel point il est dangereux de travailler en root) :

```
bash-2.05$ rm -rf ./*
bash-2.05$ rm -rf /*
```

Pour la première ligne, pas de problèmes : le contenu du répertoire courant est entièrement effacé. Pour la deuxième, ça se corse : c'est TOUT votre disque dur qui est effacé sans rémission aucune. Eh oui, pas d'undelete sous linux : on apprend à devenir des adultes responsables.

Un peu de pratique

Les 15 commandes indispensables du shell

Ce sont les 15 commandes les plus indispensables pour pouvoir travailler en mode console. Elles sont loin d'être exhaustives, mais me semblent un minimum vital à connaître pour une utilisation basique du shell.

• cat : cat affiche à l'écran le contenu d'un fichier texte. Par défaut, le flux est redirigé vers la sortie standard, c'est-à-dire vers l'écran. Mais d'autres redirections sont possibles.

```
bash-2.05$ cat toto
bash-2.05$
bash-2.05$ cat toto > /dev/null
```

• cd : cette commande signifie "change directory". Comme son nom l'indique, elle permet de passer d'un répertoire à un autre, soit par un chemin absolu, soit relatif. Le répertoire courant est le répertoire dans lequel on se trouve au moment où on en parle (ici et maintenant, quoi). Depuis le temps que j'utilise cette notion, il était temps que je la définisse, non ? Utilisée seule, la commande cd ramène l'utilisateur dans son répertoire de travail (/home/toto42 en ce qui nous concerne).

```
bash-2.05 cd /home/toto42/toto/tata/titi
bash-2.05 cd
bash-2.05 cd toto/tata/titi
```

• chmod : cette commande est d'une puissance absolument inimaginable. Vous vous souvenez des permissions ? Eh bien, chmod est la commmande qui gère ces permissions. Les arguments sont soit des lettres, soit des chiffres en mode octale. Cette seconde manière de pratiquer me semble beaucoup plus simple à utiliser une fois qu'on a mémorisé les 3 chiffres et leur signification.
L'utilisation est assez simple : chmod prend en argument -R pour la récursivité, le mode dans lequel on désire passer le ou les fichiers, ainsi que le(s) nom(s) du/des fichier(s) dont on désire changer les droits. Le mode est une suite de 3 chiffres concernant respectivement l'utilisateur, son groupe et le reste du monde. Un quatrième chiffre peut être ajouté en début de mode. Il s'agit du bit suid ou sgid, que nous avons vu un peu plus haut.

- 1 signifie que l'utilisateur concerné peut exécuter le fichier. Pour entrer dans un répertoire, il faut que celui-ci soit exécutable.
- 2 signifie que l'utilisateur concerné peut écrire dans le fichier.
- 4 signifie que l'utilisateur concerné peut lire le fichier.
- 2 en entête de ces trois chiffres signifie que le programme sera exécuté avec les droits du groupe du propriétaire et non plus de l'utilisateur. Si cela peut être très pratique, cela peut aussi être dangereux.
- 4 en entête de ces trois chiffres signifie que le programme sera exécuté avec les droits de son propriétaire (par exemple, le root). C'est ce qu'on appelle le bit SUID, et c'est la première chose que rechercheront les pirates sur un système.

La combinaison de trois chiffres vue plus haut s'utilise en les additionnant. Ainsi, 7 signifie lecture, écriture et exécution, 6 lecture et écriture, 5 lecture et exécution, et ainsi de suite. 777 signifie que n'importe qui peut faire n'importe quoi avec ce fichier. Et ça, c'est vraiment bad.

```
bash-2.05$ ls -l
total 1
drwxr-xr-x    2 toto42    users           72 Oct 10 00:35 titi
```

```
-rw-r--r--     1 toto42     users             0 Oct 10 00:33 tutu
bash-2.05$ chmod 777 titi
bash-2.05$ chmod 000 tutu
bash-2.05$ ls -l
total 1
drwxrwxrwx     2 toto42     users            72 Oct 10 00:35 titi
----------     1 toto42     users             0 Oct 10 00:33 tutu
bash-2.05$
```

- cp : cp permet de copier un répertoire ou un fichier dans un autre répertoire, ou sous un autre nom. L'option -R permet une copie récursive. Lors de la copie d'un répertoire, deux options sont possibles :

```
bash-2.05$ cp -R /home/toto42/toto /home/toto42/tutu/
bash-2.05$ cp -R /home/toto42/toto/ /home/toto42/tutu/
```

La première copie le répertoire toto dans le répertoire tutu, tandis que la seconde copie le contenu du répertoire toto dans le répertoire tutu ; ce n'est donc pas la même chose.

- find : cette commande est d'une puissance égale à sa gourmandise en ressources. Elle permet de parcourir une arborescence quelconque à la recherche d'un fichier, par son nom, ses attributs ou toute expression régulière.

```
bash-2.05$ find ./ -name toto
./toto
./coding/toto
bash-2.05$
```

- head : fort pratique, la commande head permet d'afficher les premières lignes d'un fichier. Par défaut, il s'agit des 10 premières, mais cela peut varier selon votre config.

```
bash-2.05$ head /etc/passwd
root:x:0:0::/root:/bin/tcsh
bin:x:1:1:bin:/bin:
daemon:x:2:2:daemon:/sbin:
adm:x:3:4:adm:/var/log:
lp:x:4:7:lp:/var/spool/lpd:
sync:x:5:0:sync:/sbin:/bin/sync
shutdown:x:6:0:shutdown:/sbin:/sbin/shutdown
halt:x:7:0:halt:/sbin:/sbin/halt
mail:x:8:12:mail:/:
news:x:9:13:news:/usr/lib/news:
bash-2.05$
```

- kill : c'est le grand copain de ps. Il permet d'envoyer un signal à un processus. S'il porte ce nom, c'est qu'il était, à la base, destiné à tuer le processus. Les signaux ne sont pas encore au programme, et pour nous, kill est un moyen de mettre fin, soit de manière normale, soit de manière un peu brutale, à un processus. La syntaxe est kill <option> <processus>. Les options qui nous intéressent sont -2, qui interrompt le processus en cours en lui envoyant un signal de type SIGINT, et 9, qui tue véritablement le processus en lui envoyant un signal SIGKILL. C'est très pratique en cas de plantage d'un programme, car cela vous permet de reprendre la main si une application vous lâche. Il est à noter que l'option -9 -1 termine tous les processus vous appartenant. A ne pas tester, ou alors après sauvegarde. Si vous désirez une liste complète des processus, lancez un man 7 signal.

```
bash-2.05$ ps
  PID TTY          TIME CMD
 2531 pts/0    00:00:00 tcsh
 2532 pts/0    00:00:00 bash
 2681 pts/0    00:00:00 emacs
 2730 pts/0    00:00:00 ps
bash-2.05$ kill -2 2681
bash-2.05$ ps
```

```
  PID TTY          TIME CMD
 2531 pts/0    00:00:00 tcsh
 2532 pts/0    00:00:00 bash
 2731 pts/0    00:00:00 ps
[1]+  Interrupt                 emacs
bash-2.05$
```

- ln : ln va créer un lien d'un fichier vers un autre. C'est maintenant que je dois introduire la notion très très importante de lien. Un lien est un fichier situé dans un répertoire donné, et qui va pointer vers un autre fichier existant sur le disque. Quelle utilité ? C'est bien simple : imaginons que moi, le root, j'installe un programme que je veux mettre à disposition de tous les utilisateurs, mais que, pour des raisons de sécurité, je ne veux pas le mettre dans le path. J'ai deux solutions : soit je copie ce programme par défaut, dans tous les répertoires des utilisateurs, soit je crée un lien vers ce fichier. Le lien permettra d'exécuter le fichier (ou de rentrer dans le répertoire) comme s'il se trouvait dans le répertoire courant de l'utilisateur, et l'usage de "cd .." ramènera l'utilisateur dans son répertoire de travail. De plus, un lien ne prend que quelques octets quand un programme peut prendre plusieurs mega octets. Dernier avantage et non des moindres : si les fichiers de configuration des utilisateurs sont en fait des liens pointant vers un répertoire créé par le root et où seul celui-ci a les droits en écriture, l'utilisateur ne pourra changer ces fichiers de configuration. Donc, uniformisation de la configuration des shells. Il existe deux types de liens : les liens rigides, ou "hard links" et les liens symboliques, créés avec l'option -s.

```
bash-2.05$ ln -s /home/toto42/toto toto
bash-2.05$ ln /dev/hdc /dev/cdrom
bash-2.05$ ls -l
total 1083
drwxr-xr-x    2 root     bin          1968 Nov 17 22:50 bin
drwxr-xr-x    3 root     root         1024 Nov 13 01:19 boot
drwxr-xr-x    2 root     root           48 Oct  6  1997 cdrom
drwxr-xr-x   10 root     root        47440 Nov 26 22:17 dev
drwxr-xr-x   15 root     root         2720 Nov 26 22:17 etc
drwxr-xr-x    3 root     root           72 Nov  2 16:58 home
drwxr-xr-x    3 root     root         2392 Sep 30 13:27 lib
drwxr-xr-x    6 root     root          168 Nov 20 07:59 mnt
drwxr-xr-x    4 root     root           96 Nov 29  1998 opt
dr-xr-xr-x   62 root     root            0 Nov 26 23:17 proc
drwx--x---   13 root     root          640 Nov 26 01:32 root
drwxr-xr-x    2 root     bin          2864 Sep 30 13:34 sbin
drwxrwxrwt   10 root     root          480 Nov 26 22:18 tmp
drwxr-xr-x   17 root     root          520 Jun 13 08:50 usr
drwxr-xr-x   13 root     root          384 Jun 13 08:50 var
-rw-r--r--    1 root     root      1042703 Sep 30 13:23 vmlinuz
lrwxrwxrwx    1 root     root           43 Sep 30 12:14 vmlinuz-scsi -> /usr/src
/linux-2.4.5/arch/i386/boot/bzImage
bash-2.05$ ls -l /usr/src/linux-2.4.5/arch/i386/boot/
total 1061
-rw-r--r--    1 root     root         2803 Feb 22  2001 Makefile
-rwxr-xr-x    1 root     root          512 Sep 28 20:17 bbootsect
-rw-r--r--    1 root     root         2352 Sep 28 20:17 bbootsect.o
-rw-r--r--    1 root     root         7981 Sep 28 20:17 bbootsect.s
-rw-r--r--    1 root     root         9644 Jan 30  2001 bootsect.S
-rwxr-xr-x    1 root     root         4636 Sep 30 12:08 bsetup
-rw-r--r--    1 root     root        11956 Sep 30 12:08 bsetup.o
-rw-r--r--    1 root     root        47153 Sep 30 12:08 bsetup.s
-rw-r--r--    1 root     root       910780 Sep 30 12:08 bzImage
drwxr-xr-x    2 root     root          248 Sep 30 12:08 compressed
-rw-r--r--    1 root     root          904 Jan  3  1995 install.sh
-rw-r--r--    1 root     root        24403 May  2  2001 setup.S
drwxr-xr-x    2 root     root           96 Sep 28 20:17 tools
-rw-r--r--    1 root     root        39023 Nov 21  1999 video.S
bash-2.05$
```

Dans l'exemple ci-dessus, on liste le répertoire racine, et là, on constate la présence d'un lien (reconnaissable au flag "l" dans les stats du fichier), pointant vers /usr/src/arch/i386/boot/bzImage. Ce lien fait très exactement 42 octets. Nous allons maintenant lister le répertoire en question, et nous constatons que le fichier bzImage fait en réalité 910780 octets. La différence est de taille.

- ls : ou "list". Cette commande a pour effet de lister les fichiers contenus à l'intérieur d'un répertoire. La commande ls prends de nombreuses options, mais les plus courantes sont -l m -a et -R. Ls -l affiche les répertoires et fichiers avec leurs attributs, ce qui est bien pratique pour gérer les permissions, ls -R liste récursivement un répertoire, et ls -a affiche les fichiers et répertoires cachés, notamment "." et "..". Il est d'ailleurs temps de parler de ces deux répertoires assez étranges en vérité : "." correspond au répertoire courant, et lancer un cd "." ne sert pas à grand-chose. ".." correspond, lui, au répertoire précédent dans l'arborescence, et l'invoquer permet donc de redescendre d'un niveau dans l'arborescence.

```
bash-2.05$ ls -l
total 698
-rw-r--r--    1 root     root       624517 Jun 23 00:09 System.map
-rw-r--r--    1 root     root         6128 Jun 15 07:38 boot-menu.b
-rw-r--r--    1 root     root         4368 Jun 15 07:38 boot-text.b
-rw-r--r--    1 root     root          512 Sep 30 13:45 boot.0300
lrwxrwxrwx    1 root     root           11 Sep 30 13:23 boot.b -> boot-menu.b
-rw-r--r--    1 root     root          179 Sep 30 13:45 boot_message.txt
-rw-r--r--    1 root     root          624 Jun 15 07:38 chain.b
-rw-r--r--    1 root     root        29848 Jun 23 00:09 config
drwxr-xr-x    2 root     root        12288 Sep 30 13:21 lost+found
-rw-------    1 root     root        25088 Oct  6 20:28 map
-rw-r--r--    1 root     root          652 Jun 15 07:38 os2_d.b
bash-2.05$

bash-2.05$ ls -la
total 700
drwxr-xr-x    3 root     root         1024 Oct  6 20:28 .
drwxr-xr-x   17 root     root          464 Sep 30 12:14 ..
-rw-r--r--    1 root     root       624517 Jun 23 00:09 System.map
-rw-r--r--    1 root     root         6128 Jun 15 07:38 boot-menu.b
-rw-r--r--    1 root     root         4368 Jun 15 07:38 boot-text.b
-rw-r--r--    1 root     root          512 Sep 30 13:45 boot.0300
lrwxrwxrwx    1 root     root           11 Sep 30 13:23 boot.b -> boot-menu.b
-rw-r--r--    1 root     root          179 Sep 30 13:45 boot_message.txt
-rw-r--r--    1 root     root          624 Jun 15 07:38 chain.b
-rw-r--r--    1 root     root        29848 Jun 23 00:09 config
drwxr-xr-x    2 root     root        12288 Sep 30 13:21 lost+found
-rw-------    1 root     root        25088 Oct  6 20:28 map
-rw-r--r--    1 root     root          652 Jun 15 07:38 os2_d.b
bash-2.05$
```

- man : la commande la plus utile du shell. Man affiche à l'écran la page de manuel du programme demandé. Il y en a pour tous les goûts et pour tous les programmes. Les manuels sont classés en 9 sous-sections, ce qui permet de savoir ce que l'on cherche, lorsque, par exemple, deux commandes portent le même nom. L'option -k permet une courte description des manuels. Les différentes sous-sections de man sont :
  - 1 : programmes exécutables et commandes shell.
  - 2 : appels systèmes (fonctions fournies par le noyau).
  - 3 : appels aux bibliothèques.
  - 4 : fichiers spéciaux (en général ceux de /dev).
  - 5 : format de fichiers et conventions (/etc/passwd par exemple).
  - 6 : les jeux.
  - 7 : paquetages divers.
  - 8 : administration système (pour le root essentiellement).
  - 9 : routines du noyau (pas standard).

```
bash-2.05$ man 1 write
bash-2.05$ man 2 write
bash-2.05$ man -k

bash-2.05$ mkdir tata
bash-2.05$ mkdir -p ./toto/tutu/tata/titi
```

• more : ou "plus": Cet utilitaire génial permet d'afficher le contenu d'un fichier non binaire sur la sortie standard, page par page. Son principal défaut est de ne pas pouvoir remonter à volonté le long du fichier déjà affiché.

```
bash-2.05$ more toto
  bash-2.05$
```

• mv : cette commande a deux fonctions : soit elle déplace un répertoire ou un fichier d'un point vers un autre, soit elle le renomme. Il s'agit en fait d'une seule et même action, puisque le fichier est en quelque sorte deplacé vers une nouvelle identité.

```
bash-2.05$ ls -l
total 1
drwxr-xr-x    2 toto42    users          48 Oct 10 00:33 tata
-rw-r--r--    1 toto42    users           0 Oct 10 00:32 toto
-rw-r--r--    1 toto42    users           0 Oct 10 00:33 tutu
bash-2.05$ mv toto tata/
bash-2.05$ ls -l
total 1
drwxr-xr-x    2 toto42    users          72 Oct 10 00:35 tata
-rw-r--r--    1 toto42    users           0 Oct 10 00:33 tutu
bash-2.05$ mv tata/ titi
bash-2.05$ ls -l
total 1
drwxr-xr-x    2 toto42    users          72 Oct 10 00:35 titi
-rw-r--r--    1 toto42    users           0 Oct 10 00:33 tutu
bash-2.05$
```

• ps : cette commande permet d'afficher les processus actuellement lancés sur le système, et de voir par qui ils ont été lancés, que ce soit le système ou un utilisateur.

```
bash-2.05$ ps
  PID TTY          TIME CMD
 2531 pts/0    00:00:00 tcsh
 2532 pts/0    00:00:00 bash
 2681 pts/0    00:00:00 emacs
 2683 pts/0    00:00:00 ps
bash-2.05$
```

Lancé seul, ps permet d'afficher les processus lancés sur le terminal courant. PID est le numéro du processus, TTY le terminal, TIME l'heure et CMD le nom du processus. Le PID sera fort utile pour la commande kill.

Notre second exemple va afficher à l'écran tous les processus actifs ainsi que leurs fils. Il existe de nombreuses options à ps, selon ce que nous désirons afficher à l'écran.

```
bash-2.05$ ps aux
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root         1  0.0  0.0    416    84 ?        S    Oct23   0:04 init
root         2  0.0  0.0      0     0 ?        SW   Oct23   0:00 [keventd]
root         3  0.0  0.0      0     0 ?        SW   Oct23   0:05 [kswapd]
root         4  0.0  0.0      0     0 ?        SW   Oct23   0:00 [kreclaimd]
root         5  0.0  0.0      0     0 ?        SW   Oct23   0:02 [bdflush]
root         6  0.0  0.0      0     0 ?        SW   Oct23   0:00 [kupdated]
root         7  0.0  0.0      0     0 ?        SW   Oct23   0:00 [khubd]
```

```
root          8  0.0  0.0     0     0 ?        SW<  Oct23   0:00 [mdrecoveryd]
root          9  0.0  0.0     0     0 ?        SW   Oct23   0:00 [kreiserfsd]
root         51  0.0  0.0     0     0 ?        SW   Oct23   0:00
 [kapm-idled]
bin          71  0.0  0.0  1380     4 ?        S    Oct23   0:00 /sbin/rpc.portmap
root         77  0.0  0.0  1784   228 ?        S    Oct23   0:00 /usr/sbin/syslogd
root         80  0.0  0.1  1928   408 ?        S    Oct23   0:00 /usr/sbin/klogd -
root         82  0.0  0.0  1216     4 ?        S    Oct23   0:00 /usr/sbin/inetd
root         85  0.0  0.0  2624   132 ?        S    Oct23   0:00 /usr/sbin/sshd
root         88  0.0  0.0  1356   304 ?        S    Oct23   0:00 /usr/sbin/crond -
daemon       90  0.0  0.0  1356    56 ?        S    Oct23   0:00 /usr/sbin/atd -b
root         97  0.0  0.1  2896   532 ?        S    Oct23   0:00 sendmail: accepti
root        101  0.0  0.0  1220     4 ?        S    Oct23   0:00 /usr/sbin/apmd
root        106  0.0  0.4 40788 1648 ?        S    Oct23   0:00 /usr/sbin/httpd
root        108  0.0  0.0  1260    80 ?        S    Oct23   0:00 gpm -m /dev/mouse
root        111  0.0  0.0  1220     4 tty2     S    Oct23   0:00 /sbin/agetty 3840
root        112  0.0  0.0  1220     4 tty3     S    Oct23   0:00 /sbin/agetty 3840
root        113  0.0  0.0  1220     4 tty4     S    Oct23   0:00 /sbin/agetty 3840
root        114  0.0  0.0  1220     4 tty5     S    Oct23   0:00 /sbin/agetty 3840
root        115  0.0  0.0  1220     4 tty6     S    Oct23   0:00 /sbin/agetty 3840
toto42      116  0.0  0.2  2780  1036 tty8     S    Oct23   0:00 -tcsh
root        117  0.0  0.0  1220     4 tty9     S    Oct23   0:00 /sbin/agetty 3840
root        118  0.0  0.0  1220     4 tty10    S    Oct23   0:00 /sbin/agetty 3840
root        119  0.0  0.0  1220     4 tty11    S    Oct23   0:00 /sbin/agetty 3840
nobody      121  0.0  0.5 40884  2192 ?        S    Oct23   0:00 /usr/sbin/httpd
nobody      122  0.0  0.5 40900  2300 ?        S    Oct23   0:00 /usr/sbin/httpd
nobody      123  0.0  0.4 40880  1912 ?        S    Oct23   0:00 /usr/sbin/httpd
nobody      124  0.0  0.5 40880  2196 ?        S    Oct23   0:00 /usr/sbin/httpd
nobody      125  0.0  0.5 40884  2192 ?        S    Oct23   0:00 /usr/sbin/httpd
nobody      685  0.0  0.5 40884  2176 ?        S    Oct23   0:00 /usr/sbin/httpd
nobody      687  0.0  0.5 40904  2140 ?        S    Oct23   0:00 /usr/sbin/httpd
nobody      688  0.0  0.5 40880  2188 ?        S    Oct23   0:00 /usr/sbin/httpd
nobody      689  0.0  0.5 40884  2184 ?        S    Oct23   0:00 /usr/sbin/httpd
nobody      690  0.0  0.5 40888  2180 ?        S    Oct23   0:00 /usr/sbin/httpd
toto42     1294  0.0  0.4  2780  1544 tty1     S    Oct24   0:00 -tcsh
toto42     2473  0.0  0.2  1792   908 tty8     S    Oct25   0:00 /bin/sh /usr/X11R
toto42     2480  0.0  0.1  2152   688 tty8     S    Oct25   0:00 xinit /home/toto42
root       2481  0.6  3.6 58972 14188 ?        S    Oct25   0:33 X :0
toto42     2501  0.0  0.6  4160  2452 tty8     S    Oct25   0:02 /usr/X11R6/bin/wm
toto42     2506  0.0  1.7 19528  6592 tty8     S    Oct25   0:00 xmms
toto42     2507  0.0  1.7 19528  6592 tty8     S    Oct25   0:00 xmms
toto42     2508  0.0  1.7 19528  6592 tty8     S    Oct25   0:00 xmms
toto42     2509  0.0  1.7 19528  6592 tty8     S    Oct25   0:00 xmms
toto42     2510  0.0  0.6  4532  2432 tty8     S    Oct25   0:00 aumix
toto42     2528  0.3  1.4  7380  5424 tty8     S    Oct25   0:16 emacs
root       2530  0.0  0.5  4184  2212 tty8     S    Oct25   0:01 xterm -sb
toto42     2531  0.0  0.3  2448  1256 pts/0    S    Oct25   0:00 -csh
toto42     2532  0.0  0.3  2008  1208 pts/0    S    Oct25   0:00 bash
root       2577  0.0  0.1  1220   504 tty12    S    Oct25   0:00 /sbin/agetty 3840
toto42     2681  0.1  1.2  7044  4756 pts/0    S    Oct25   0:00 emacs
toto42     2690  0.3  1.7 19528  6592 tty8     S    00:03   0:00 xmms
toto42     2691  0.3  1.7 19528  6592 tty8     S    00:03   0:00 xmms
toto42     2692  0.0  0.2  2560   828 pts/0    R    00:04   0:00 ps aux
bash-2.05$
```

Note : *au vu du nombre de processus lancés, de la faible valeur du PID et de l'âge du capitaine, on peut assez facilement déduire qu'il s'agit d'une machine lancée depuis très longtemps, ou alors très utilisée. La date du lancement d'INIT, premier démon à se lancer au démarrage, nous indique deux jours d'uptime. On peut donc en conclure qu'il s'agit d'une machine très utilisée. Un petit rien comme ça permet aux pirates de mettre au point leurs techniques d'échappatoire.*

- USER indique l'utilisateur ayant lancé le processus. Vous n'avez un pouvoir que sur les processus que vous avez vous-même lancés, sinon ce serait trop facile.
- PID est le numéro du processus. Ce nombre est fort utile pour agir sur le processus en question. Vous pouvez remarquer que les numéros de processus ne se suivent pas forcément.
- %CPU est la charge que le processus fait peser sur le processeur. Dans la plupart des cas, cette charge sera à 0%, sauf si l'utilisateur utilise ce processus. La commande top est similaire à ps si ce n'est qu'elle affiche les processus en temps réel.
- %mem est la charge du processus sur la mémoire vive.
- TTY est le numéro du terminal sur lequel le processus é été lancé. Le numéro de ce tty permet de savoir un très grand nombre de choses sur l'utilisateur qui gère le processus. Par exemple, pts/0 signifie que le processus a été lancé par un utilisateur connecté à distance, ou ayant lancé un xterm.
- STATE est l'état actuel du processus (actif, en sommeil ou zombi).
- DATE est la date de lancement du processus.
- TIME est son heure d'exécution.
- COMMAND est la commande utilisée pour lancer le processus.
- pwd : la commande pwd permet d'afficher le répertoire courant sur la sortie standard, le chemin étant indiqué de manière absolue.

```
bash-2.05$ pwd
/usr/local/man/man4
bash-2.05$
```

- rm : cette commande sert à supprimer un fichier. Les options les plus courantes sont -r, qui élimine des fichiers récursivement, et -f qui force la suppression des fichiers. L'option -f s'utilise essentiellement quand les fichiers à effacer sont des répertoires. En effet, la commande rm ne peut, en temps normal, effacer des répertoires. Quoi qu'il en soit, il faut être d'une prudence extrême quant à l'utilisation de ces options, car un simple espace en trop et c'est la catastrophe.

```
bash-2.05$ rm toto
bash-2.05$ rm -rf /*
```

- rmdir : ou "remove directory" en bon anglais. Cette commande permet de supprimer un répertoire vide. Si celui-ci ne l'est pas, un message d'erreur est envoyé sur la sortie standard.

```
bash-2.05$ rmdir toto
bash-2.05$ rmdir /home/toto
bash-2.05$ rmdir tutu
rmdir: `tutu': Directory not empty
bash-2.05$
```

- tail : tail ressemble pas mal à head, mais affiche les 10 dernières lignes d'un fichier. C'est avec l'option -f que cette commande prend toute sa puissance, puisque tail va attendre que la fin du fichier soit modifiée avant d'afficher les modifications à l'écran en temps réel. C'est très pratique pour afficher les logs d'un firewall, entre autres. C'est une commande particulièrement utile en administration système. Elle se termine avec un ^C.

```
bash-2.05$ tail /etc/passwd
mail:x:8:12:mail:/:
news:x:9:13:news:/usr/lib/news:
uucp:x:10:14:uucp:/var/spool/uucppublic:
operator:x:11:0:operator:/root:/bin/bash
games:x:12:100:games:/usr/games:
ftp:x:14:50::/home/ftp:
mysql:x:27:27:MySQL:/var/lib/mysql:/bin/bash
gdm:x:42:42:GDM:/var/state/gdm:/bin/bash
nobody:x:99:99:nobody:/:
toto42:x:100:100::/home/toto42/:/bin/tcsh
bash-2.05$
```

• touch : utilisée telle quelle, cette commande permet de créer un fichier de taille 0 octets à l'endroit désiré, avec le nom désiré. Les droits sur ce fichier sont ceux indiqués par la commande umask. Touch peut prendre plusieurs options, mais la plus utile reste sans doute la commande -t, qui permet de changer la date et l'heure de la dernière modification d'un fichier.

```
bash-2.05$ touch toto
bash-2.05$ ls -l toto
-rw-r--r--    1 toto42    users        0 Oct 10 00:18 toto
bash-2.05$ touch -t 200112250000 toto
bash-2.05$ ls -l toto
-rw-r--r--    1 toto42    users        0 Dec 25  2001 toto
bash-2.05$
```

Comprendre la console

La plupart des utilitaires présentés ici concernent la manipulation de fichiers textes ou de chaînes de caractères. Dans un environnement où ces fichiers sont très nombreux, des utilitaires et outils puissants sont un besoin impérieux qui permettent de transformer, ce qui pourrait passer pour une lourdeur en un atout redoutable. Nous allons d'abord étudier certains procédés avant de nous pencher de manière très succinte sur des utilitaires et des commandes qui permettent de transformer la console en un outil phénoménalement puissant.

Les redirections

Sous linux, la plupart des entrées sorties se font sous forme de flux, canalisés vers l'une ou l'autre sortie. Tout comme il est possible de détourner un fleuve, il est possible, et même pratique, de détourner ces flux de données pour agir dessus en temps réel. Il est par exemple fort utile de rediriger les informations qui s'affichent à l'écran vers un fichier texte, ou vers un périphérique spécial dans le cas par exemple des utilisateurs non voyants. Ces redirections sont effectuées à l'aide de ce que l'on appelle les opérateurs de redirection.

• L'opérateur '>' : il permet de rediriger un flux vers la sortie de son choix. La sortie est créée, et si elle existe déjà, elle sera créée, sauf dans un cas très particulier qui est /dev/null. /dev/null est un périphérique extrêmement utile, que l'on pourrait comparer au néant : c'est un trou sans fond, et ce qui y est redirigé y est donc englouti. Une sorte de tonneau des danaïdes version unix, quoi.

```
bash-2.05$ ls -l
total 30
drwxr-xr-x    3 toto42    users         688 Oct 22 20:50 -img
drwxr-xr-x    2 toto42    users         176 Oct 23 00:06 -lib
-rw-r--r--    1 toto42    users           0 Oct 23 00:09 404.php
-rw-r--r--    1 toto42    users           0 Oct 22 21:19 code.php
-rw-r--r--    1 toto42    users        6461 Oct 22 23:15 contact.php
-rw-r--r--    1 toto42    users        2669 Oct 22 23:06 index.php
-rw-r--r--    1 toto42    users           0 Oct 22 21:14 links.php
-rw-r--r--    1 toto42    users        1415 Oct 23 20:07 main.php
-rw-r--r--    1 toto42    users        2668 Oct 22 20:48 maquette.php
drwxr-xr-x    2 toto42    users          72 Oct 23 00:09 misc
-rw-r--r--    1 toto42    users        1008 Oct 22 23:29 misc.html
-rw-r--r--    1 toto42    users         883 Oct 23 00:06 misc.php
-rw-r--r--
    1 toto42    users              0 Oct 22 21:14 unix.php
bash-2.05$ ls -l > toto
```

Rien ne s'affiche car le flux a été redirigé vers le fichier toto au lieu de la sortie standard.

```
bash-2.05$ ls -l
total 34
```

```
drwxr-xr-x    3 toto42    users         688 Oct 22 20:50 -img
drwxr-xr-x    2 toto42    users         176 Oct 23 00:06 -lib
-rw-r--r--    1 toto42    users           0 Oct 23 00:09 404.php
-rw-r--r--    1 toto42    users           0 Oct 22 21:19 code.php
-rw-r--r--    1 toto42    users        6461 Oct 22 23:15 contact.php
-rw-r--r--    1 toto42    users        2669 Oct 22 23:06 index.php
-rw-r--r--    1 toto42    users           0 Oct 22 21:14 links.php
-rw-r--r--    1 toto42    users        1415 Oct 23 20:07 main.php
-rw-r--r--    1 toto42    users        2668 Oct 22 20:48 maquette.php
drwxr-xr-x    2 toto42    users          72 Oct 23 00:09 misc
-rw-r--r--    1 toto42    users        1008 Oct 22 23:29 misc.html
-rw-r--r--    1 toto42    users         883 Oct 23 00:06 misc.php
-rw-r--r--    1 toto42    users         912 Oct 23 21:32 toto
-rw-r--r--    1 toto42    users           0 Oct 22 21:14 unix.php
```

On s'aperçoit que le fichier toto qui n'existait pas lors de notre premier ls a maintenant été créé.

```
bash-2.05$ cat toto
total 30
drwxr-xr-x    3 toto42    users         688 Oct 22 20:50 -img
drwxr-xr-x    2 toto42    users         176 Oct 23 00:06 -lib
-rw-r--r--    1 toto42    users           0 Oct 23 00:09 404.php
-rw-r--r--    1 toto42    users           0 Oct 22 21:19 code.php
-rw-r--r--    1 toto42    users        6461 Oct 22 23:15 contact.php
-rw-r--r--    1 toto42    users        2669 Oct 22 23:06 index.php
-rw-r--r--    1 toto42    users           0 Oct 22 21:14 links.php
-rw-r--r--    1 toto42    users        1415 Oct 23 20:07 main.php
-rw-r--r--    1 toto42    users        2668 Oct 22 20:48 maquette.php
drwxr-xr-x    2 toto42    users          72 Oct 23 00:09 misc
-rw-r--r--    1 toto42    users        1008 Oct 22 23:29 misc.html
-rw-r--r--    1 toto42    users         883 Oct 23 00:06 misc.php
-rw-r--r--    1 toto42    users           0 Oct 23 21:32 toto
-rw-r--r--    1 toto42    users           0 Oct 22 21:14 unix.php
bash-2.05$
```

Le contenu du fichier toto est absolument identique à ce qu'avait donné notre premier ls, à une différence près : lorsque nous lançons notre commande, le fichier toto est immédiatement créé, mais il ne contient rien quand la commande est lancée. Lorsqu'il est listé, il fait donc 0 octets, et, à la fin du processus de listing, il fait 912 octets.

- L'opérateur '>>': il a la même fonction que l'opérateur '>' si ce n'est que lorsque la sortie indiquée existe déjà, les données sont ajoutées à la suite de celles existant déjà, au lieu que celles-ci soient écrasées. Si la sortie n'existe pas, elle est créée.

```
bash-2.05$ cat toto
total 30
drwxr-xr-x    3 toto42    users         688 Oct 22 20:50 -img
drwxr-xr-x    2 toto42    users         176 Oct 23 00:06 -lib
-rw-r--r--    1 toto42    users           0 Oct 23 00:09 404.php
-rw-r--r--    1 toto42    users           0 Oct 22 21:19 code.php
-rw-r--r--    1 toto42    users        6461 Oct 22 23:15 contact.php
-rw-r--r--    1 toto42    users        2669 Oct 22 23:06 index.php
-rw-r--r--    1 toto42    users           0 Oct 22 21:14 links.php
-rw-r--r--    1 toto42    users        1415 Oct 23 20:07 main.php
-rw-r--r--    1 toto42    users        2668 Oct 22 20:48 maquette.php
drwxr-xr-x    2 toto42    users          72 Oct 23 00:09 misc
-rw-r--r--    1 toto42    users        1008 Oct 22 23:29 misc.html
-rw-r--r--    1 toto42    users         883 Oct 23 00:06 misc.php
-rw-r--r--    1 toto42    users           0 Oct 23 21:32 toto
-rw-r--r--    1 toto42    users           0 Oct 22 21:14 unix.php
```

```
bash-2.05$ ls -l >> toto
```

On s'aperçoit que la sortie standard a bien été redirigée vers notre fichier toto.

```
bash-2.05$ cat toto
total 30
drwxr-xr-x    3 toto42    users         688 Oct 22 20:50 -img
drwxr-xr-x    2 toto42    users         176 Oct 23 00:06 -lib
-rw-r--r--    1 toto42    users           0 Oct 23 00:09 404.php
-rw-r--r--    1 toto42    users           0 Oct 22 21:19 code.php
-rw-r--r--    1 toto42    users        6461 Oct 22 23:15 contact.php
-rw-r--r--    1 toto42    users        2669 Oct 22 23:06 index.php
-rw-r--r--    1 toto42    users           0 Oct 22 21:14 links.php
-rw-r--r--    1 toto42    users        1415 Oct 23 20:07 main.php
-rw-r--r--    1 toto42    users        2668 Oct 22 20:48 maquette.php
drwxr-xr-x    2 toto42    users          72 Oct 23 00:09 misc
-rw-r--r--    1 toto42    users        1008 Oct 22 23:29 misc.html
-rw-r--r--    1 toto42    users         883 Oct 23 00:06 misc.php
-rw-r--r--    1 toto42    users           0 Oct 23 21:32 toto
-rw-r--r--    1 toto42    users           0 Oct 22 21:14 unix.php
total 34
drwxr-xr-x    3 toto42    users         688 Oct 22 20:50 -img
drwxr-xr-x    2 toto42    users         176 Oct 23 00:06 -lib
-rw-r--r--    1 toto42    users           0 Oct 23 00:09 404.php
-rw-r--r--    1 toto42    users           0 Oct 22 21:19 code.php
-rw-r--r--    1 toto42    users        6461 Oct 22 23:15 contact.php
-rw-r--r--    1 toto42    users        2669 Oct 22 23:06 index.php
-rw-r--r--    1 toto42    users           0 Oct 22 21:14 links.php
-rw-r--r--    1 toto42    users        1415 Oct 23 20:07 main.php
-rw-r--r--    1 toto42    users        2668 Oct 22 20:48 maquette.php
drwxr-xr-x    2 toto42    users          72 Oct 23 00:09 misc
-rw-r--r--    1 toto42    users        1008 Oct 22 23:29 misc.html
-rw-r--r--    1 toto42    users         883 Oct 23 00:06 misc.php
-rw-r--r--    1 toto42    users         912 Oct 23 21:32 toto
-rw-r--r--    1 toto42    users           0 Oct 22 21:14 unix.php
bash-2.05$
```

Le listage de notre répertoire s'est ajouté à la suite des données précédemment entrées. Le fichier toto n'a pas été écrasé.

- L'opérateur '<<': il bloque le processus tant que les caractères situés juste après n'ont pas été tapés suivis de la touche entrée.

```
bash-2.05$ ls << toto > tutu
> tutu
> titi
> toto
bash-2.05$ cat tutu
-img
-lib
404.php
code.php
contact.php
index.php
links.php
main.php
maquette.php
misc
misc.html
misc.php
toto
```

```
tutu
unix.php
bash-2.05$
```

- L'opérateur 'l': ou pipe. C'est sans doute le plus puissant des opérateurs de redirection. En effet, il permet de rediriger le flux de sortie d'une commande en entrée d'une autre commande. On peut ainsi faire des lignes de commande contenant une dizaine de pipes et faisant des trucs hallucinants une fois qu'on sait se servir de certains utilitaires de formatage de chaînes de caractères. La notion de pipes semble souvent un peu confuse, mais un exemple simple permet de mieux comprendre.

```
bash-2.05$ cat toto | grep tu
-rw-r--r--    1 toto42    users       124 Oct 23 21:45 tutu
-rw-r--r--    1 toto42    users       124 Oct 23 21:45 tutu
bash-2.05$
```

grep est un utilitaire permettant de faire une recherche de correspondance dans un fichier ou une chaîne de caractères. Il nous a permis d'afficher le contenu de notre fichier toto, mais en le filtrant au travers de l'utilitaire grep de manière à n'avoir en sortie que les lignes contenant la chaîne de caractères "tu".

Les caractères de contrôle

- Le ';': dans une chaîne d'instructions placées sur la même ligne, le caractère ',' permet de séparer les instructions les unes des autres. Il n'y a aucune relation entre chacune des commandes lancées, et il s'agit simplement d'une liste de commandes à exécuter. Il n'y a pas de contrôle de succès de la première commande avant de passer à la seconde.

```
bash-2.05$ rm *.exe ; echo "toto"
rm: cannot remove `*.exe': No such file or directory
toto
bash-2.05$
```

Même s'il n'existe pas de fichier .exe dans le répertoire courant, la seconde instruction est quand même lancée (le contraire aurait d'ailleurs été un peu bizarre sur un PC n'ayant pas vu un fichier .exe depuis.... hou la la, au moins tout ça).

- Le '&&': le double "et commercial" se place entre deux commandes dans une ligne où l'on désire exécuter plusieurs instructions. Contrairement au ';', le '&&' vérifie le succès d'une tâche avant de passer à la suivante, sinon la ligne de commande s'interrompt et un message est affiché sur la sortie d'erreur.

```
bash-2.05$ rm *.exe && echo "toto"
rm: cannot remove `*.exe': No such file or directory
bash-2.05$
```

- Le '&': il permet de lancer un processus, de le passer en tâche de fond avant de rendre la main à l'utilisateur. La tâche peut cependant être rappelée par la suite, grâce à la commande fg, comme foreground.

```
bash-2.05$ emacs &
[1] 673
bash-2.05$
```

Les utilitaires ultimes

Ils sont nombreux dans les environnnements *nix les utilitaires très puissants et directement intégrés au shell. S'il est possible d'utiliser linux sans jamais voir une seule ligne de commande, ce serait vraiment très très dommage et ce

serait passer à côté de choses VRAIMENT ultimes, notamment grâce aux pipes. Les explications et exemples donnés ci-dessous ne dispensent pas de consulter vos deux meilleurs amis : google et le man. Ils sont en effet une mine de renseignements incomparables dès que vous avez besoin de savoir quelque chose.
Tous les utilitaires dont je vais vous parler ici ne sont pas directement intégrés au shell mais on les trouve sur toutes les distributions et ils sont vraiment utiles ; c'est pour cela qu'il m'a semblé important de les décrire.

- grep : avec son cousin egrep, ce sont des outils permettant de faire ce que l'on appelle du "pattern matching", c'est-à-dire une recherche de correspondance au sein soit d'une chaîne, soit d'une expression régulière. Cette seconde possibilité donne alors toute sa puissance à grep. Je vous conseille fortement de faire un man regexp pour étudier les expressions régulières car elles ne sont pas étudiées dans ce cours.

```
bash-2.05$ cat /etc/passwd | grep toto42
toto42:x:100:100::/home/toto42/:/bin/tcsh
```

On fait une recherche sur toutes les chaînes contenant l'occurence 'toto42'

```
bash-2.05$ cat /etc/passwd | grep ^n
news:x:9:13:news:/usr/lib/news:
nobody:x:99:99:nobody:/:
bash-2.05$
```

On fait ici une recherche dans le fichier /etc/passwd sur toutes les chaînes commençant par un 'n'.

```
bash-2.05$ grep sock *.c
xaccept.c:#include <sys/socket.h>
xaccept.c:int            xaccept(int sockfd, struct sockaddr *addr, int *addrlen)
xaccept.c:  len = sizeof(struct sockaddr);
xaccept.c:  if ((new_fd = accept(sockfd, addr, &len)) == -1)
xbind.c:#include <sys/socket.h>
xbind.c:#include "libsocks.h"
xbind.c:int      xbind(int sockfd, struct sockaddr *my_addr, int addr_len)
xbind.c:  if ((bind(sockfd, (struct sockaddr *)&my_addr, sizeof(struct sockaddr)
)) == -1)
xconnect.c:#include <sys/socket.h>
xconnect.c:#include "libsocks.h"
xconnect.c:int  xconnect(int sockfd, struct sockaddr *serv_addr, int addr_len)
xconnect.c:  if ((connect(sockfd, (struct sockaddr *)&serv_addr, sizeof (struct
sockaddr))) == -1)
xlisten.c:#include <sys/socket.h>
xlisten.c:#include "libsocks.h"
xlisten.c:int    xlisten(int sockfd, int backlog)
xlisten.c:  if ((listen(sockfd, backlog)) == -1)
xsocket.c:#include <sys/socket.h>
xsocket.c:#include "libsocks.h"
xsocket.c:int    xsocket(int domain, int type, int protocole)
xsocket.c:  int sock;
xsocket.c:  if ((sock = socket(domain, type, protocole)) == -1)
xsocket.c:      printf("Error: socket\n");
xsocket.c:    return (sock);
bash-2.05$
```

Dans ce dernier exemple, on utilise directement grep sur tous les fichiers c à la recherche de la string "sock". Grep nous affiche alors les lignes correspondantes avec le fichier dont elles ont été extraites.

- cut : cet utilitaire permet de faire des coupes franches dans des fichiers texte, par exemple pour afficher tous les champs d'une liste située entre deux délimiteurs. Combiné à grep et à d'autres utilitaires du même genre, via les pipes, cet utilitaire permet de faire des trucs de fou (comme tout ce qu'on peut faire en console en fait).

Dans notre exemple, nous allons faire une recherche sur le fichier /etc/passwd, sur les chaînes commençant par n, mais en ne retournant que leur UID et le nom des utilisateurs. La ligne étant un peu compliquée (très simple en fait :-), je vais vous la décrire pas à pas.

"cat /etc/passwd" parcourt le fihier /etc/passwd.

"grep ^n" fait une recherche sur les lignes commençant par 'n'.

"cut -f 3,5" prend les champs 3 et 5.

"-d :" déclare que ces champs sont séparés par le délimiteur ':'.

```
bash-2.05$ cat /etc/passwd | grep ^n | cut -f 3,5 -d : > toto
bash-2.05$ cat toto
9:news
99:nobody
100:
bash-2.05$
```

- sed : cet utilitaire fait tout sauf le café, ou presque. Nous l'utiliserons ici pour remplacer des occurences dans des chaînes de caractères, ce qui n'est qu'une de ses très nombreuses fonctionnalités. En fait, pour dire tout ce que fait sed, il faudrait un livre entier (il existe : Sed et hawk, éditions o'reilly(9)).

Notre exemple va consister à parcourir le fichier /etc/passwd, extraire les logins UIDs et GIDs des utilisateurs dont le nom commence par 'n', d'afficher ces informations, mais en remplaçant le separateur ':' par des ',', puis, de tout rediriger vers notre fichier.... toto (original n'est-ce pas ?). Une fois de plus, c'est un peu compliqué, donc je vais commmenter pas à pas. Jusqu'à sed, pas de problème, on l'a déjà vu.

sed -e va appliquer sed sur une expression

's/:/,/' remplace toutes les occurences de ':' par ','. L'option -s aurait appliqué ce changement sur la première occurence remplacée.

```
bash-2.05$ cat /etc/passwd | grep ^n | cut -f 1,3,4 -d : | sed -e\
's/:/,/' > toto
bash-2.05$ cat toto
news,9,13
nobody,99,99
bash-2.05$
```

- sort : permet de trier les lignes d'un fichier par ordre alphabétique. L'option -n trie dans le bon ordre l'option -r dans l'ordre inverse.

```
bash-2.05$ sort -n /etc/passwd
adm:x:3:4:adm:/var/log:
bin:x:1:1:bin:/bin:
daemon:x:2:2:daemon:/sbin:
ftp:x:14:50::/home/ftp:
games:x:12:100:games:/usr/games:
gdm:x:42:42:GDM:/var/state/gdm:/bin/bash
halt:x:7:0:halt:/sbin:/sbin/halt
lp:x:4:7:lp:/var/spool/lpd:
mail:x:8:12:mail:/:
mysql:x:27:27:MySQL:/var/lib/mysql:/bin/bash
news:x:9:13:news:/usr/lib/news:
nobody:x:99:99:nobody:/:
operator:x:11:0:operator:/root:/bin/bash
root:x:0:0::/root:/bin/tcsh
shutdown:x:6:0:shutdown:/sbin:/sbin/shutdown
sync:x:5:0:sync:/sbin:/bin/sync
uucp:x:10:14:uucp:/var/spool/uucppublic:
```

- tr : dans le genre utilitaire utile, tr se pose là : il sert à remplacer toutes les occurences d'un caractère affichable par une autre. Si cela peut être parfaitement inutile (comme le montrera notre exemple 1), il peut aussi être très pratique lorsqu'on travaille sur de

```
Report bugs to <bug-make@gnu.org>.
```

gros fichiers.

```
bash-2.05$ echo "Je suis un linuxien d'elite, le pingouin c'est\ bien, linux aussi" | tr eslota
351074
J3 5ui5 un linuxi3n d'31i73, 13 ping0uin c'357 bi3n, linux 4u55i
bash-2.05$
```

Voilà, vous savez maintenant écrire en warlordz sans vous fatiguer. Complètement inutile, donc complètement indispensable !!! Mais voyons quelque chose d'un peu plus sérieux maintenant.

```
bash-2.05$ cat /etc/passwd | grep ^n | cut -f 1,3,4 -d : | sed -e\
'y/:/,/' | tr n N> toto
bash-2.05$ more toto
News,9,13
Nobody,99,99
bash-2.05$
```

Voilà, nous avons repris l'exemple vu plus haut, mais en mettant les n en majuscule, parce que ça fait plus propre.

- wc : word count est un utilitaire bien pratique qui permet de compter le nombre de lignes, de mots et de lettres d'un fichier texte. Ça peut sembler un peu inutile, mais vous comprendrez très vite que ça peut se montrer bien utile au contraire.
  wc -w compte le nombre de mots dans un fichier.
  wc -l compte le nombre de lignes.
  wc -L donne la longueur de la plus longue ligne.
  wc -c compte le nombre de caractères.

```
bash-2.05$ wc /etc/passwd
     18      18     579 /etc/passwd
bash-2.05$
```

Mon fichier /etc/passwd contient 18 lignes, 18 mots et 579 caractères. Les mots sont comptés à chaque espace, c'est pour cela que wc en compte 18.

Voilà pour ce paragraphe. Je vous propose maintenant un petit exercice pour mettre en application tout ce que nous avons vu : sur une seule ligne de commande, et uniquement à l'aide de pipes et d'un seul ';' (donc pas le droit à && :) vous allez créer un fichier recensant tous les fichiers .gif et .jpg présents sur votre disque dur, en donnant leur nom, leur propriétaire (nom et groupe), et leur taille, classés par ordre alphabétique. La dernière ligne du fichier sera le nombre de ces images contenues dans le fichier, et le nom de chacune de ces images devra commencer par une majuscule. Bonne chance, c'est un exercice des plus difficiles.

Sécuriser Linux

I. Sécurisation de base

Même si la station Linux est réputée plus sécurisée que d'autres environnements de type Win32, il convient de procéder à quelques modifications pour que la sécurité de votre poste et de vos données soit optimum. Découvrez comment rendre encore plus " secure " le plus sûr des systèmes.

Nous allons voir les principes de base pour rendre son poste de travail GNU/Linux encore plus sécurisé. Notre démarche se basera sur le fait que vous possédez une distribution déjà installée et que celle-ci n'est pas compromise. En cas de doute, je ne saurais trop vous conseiller de faire une réinstallation complète de votre système, en ayant, bien sûr, auparavant sauvegardé vos données personnelles. Ceci étant dit, à vos claviers !

Sécurité physique

Le BIOS (Basic Input/Output System). Une chose que vous pouvez faire (en dehors de Linux) pour rendre votre machine plus sûre est d'activer la protection par mot de passe de votre BIOS. En effet, quelque soit la politique de sécurité appliquée au niveau de votre système, il est possible à une personne physique de couper l'alimentation de votre machine et de redémarrer votre ordinateur à partir d'une disquette par exemple. Il est assez facile pour quelqu'un maîtrisant un minimum Linux de passer des paramètres de démarrage au noyau qui peuvent conduire à l'accès complet à vos données (voir LILO). En revanche, si vous avez mis en place une protection au niveau du BIOS, le redémarrage de la machine sera impossible sans votre mot de passe.

Cette démarche constitue donc la première étape pour la sécurisation de votre système. La première chose est de s'assurer qu'aucune autre personne ne pourra changer les paramètres du BIOS et que le mot de passe soit obligatoire au démarrage de la machine. Pour activer ces protections, il suffit simplement de se rendre dans les paramètres de configurations de votre BIOS. Il n'y a pas à proprement parler une technique qui permet d'y arriver tant le nombre de configuration est diversifié. Pour y accéder, il suffit d'appuyer sur la touche correspondante au message d'accueil inscrit au démarrage de votre machine :

```
DEL to enter SETUP,
SUPPR to enter SETUP,
INS to enter SETUP,
F1 to enter SETUP,
ALT+F1 for SETUP, ALT+F2 for FLASH,
```

jusqu'à l'apparition du menu de configuration de votre BIOS. La configuration des mots de passe pour le changement des données du BIOS ou le démarrage se font généralement dans les sous-menus : BIOS Setup, Security Setup ou General Setup. Une fois les mots de passe entrés, sauvegardez votre BIOS (touche F10) et redémarrez. La protection par mot de passe est activée. Attention de ne pas les oublier ! Sans eux, vous n'aurez plus la possibilité de démarrer votre box ;).

Processus d'amorçage

LILO (Linux LOader). LILO est un programme qui se charge de lancer le noyau Linux lorsque le BIOS a passé la main au système d'exploitation de la machine. Il est installé par défaut avec toutes les distributions récentes. Si vous souhaitez vérifier l'existence de LILO sur votre machine, maintenez la touche Majuscule enfoncée lors du boot sur votre disque dur au démarrage de la machine. Si vous voyez apparaître cette invite : LILO boot: alors c'est que vous êtes potentiellement vulnérable au type d'attaque que nous décrivons ci-dessous. Dans le cas contraire, ne croyez pas que cela soit impossible et lisez tout de même ce qui suit.

Une des fonctionnalités offerte par LILO est la possibilité de passer des arguments au noyau Linux lors du démarrage de celui-ci. Cette option peut se révéler particulièrement utile si vous avez besoin de procéder à des opérations de maintenance ou s'il est impossible d'amorcer le système de manière normale (suite à des coupures de courant ou des pertes de données par exemple). Pour démarrer Linux en mode maintenance, il suffit d'indiquer l'argument " single " à l'invite boot :

```
LILO boot: linux single
```

Cet argument charge le noyau linux dans un mode d'administration, accessible uniquement au root. Le danger ici est que le fonctionnement normal du système, tel que vous l'aviez défini, n'est plus valable et qu'à la place il reste simplement une invite login root. Mais plus terrible encore est l'argument " init ". Celui-ci permet de charger un système linux sans aucune de ses fonctionnalités, services et paramètres. C'est-à-dire que n'importe qui passant cette argument se voit attribué un shell root sur la machine sans qu'aucun mot de passe ne lui ait été demandé :

```
LILO boot: linux init=/bin/bash
```

Il paraît donc évident que des mesures pour sécuriser LILO doivent être prises rapidement ! Je ne rentrerai pas dans une explication détaillée du fichier de configuration /etc/lilo.conf car ce n'est pas l'objectif de cet article. Je vous donne simplement les lignes à ajouter à celui-ci pour le rendre un peu plus sécurisé. Pour éditer et modifier ce fichier, vous devez être root (la commande su fait très bien l'affaire) :

```
[user@bdm ~]$ su
Password:
[root@bdm ~]# vi /etc/lilo.conf
```

Pour s'assurer que personne ne pourra passer d'arguments au noyau sans mot de passe, il suffit d'ajouter les deux lignes en gras dans le fichier de configuration :

```
# exemple de fichier de configuration /etc/lilo.conf
boot=/dev/hda
map=/boot/map
default=linux
prompt
timeout=200
password="insérez_votre_mot_de_passe_ici"
restricted
image=/boot/vmlinuz
        label=linux
        root=/dev/hdb1
        initrd=/boot/initrd.img
        append="quiet devfs=mount hdd=ide-scsi"
        vga=normal
        read-only
other=/dev/hda1
        label=NT
        table=/dev/hda
```

Cette configuration permettra à toutes les images présentes dans le fichier d'être amorçables sans mot de passe à moins que des arguments n'aient été fournis au noyau. Si un utilisateur souhaitait passer des arguments au noyau, celui-ci devrait d'abord saisir le mot de passe associé. Pour enregistrer vos modifications et que celles-ci soient actives au prochain redémarrage, vous devez taper à l'invite de commande (toujours en root) : lilo. La nouvelle configuration est alors sauvegardée. Pour finir avec LILO, il convient de changer les per-

missions d'accès en lecture sur le fichier. Et oui, sinon tout le monde peut éditer celui-ci et y lire en clair votre mot de passe et passer à nouveau des paramètres au noyau. Pour cela, il suffit d'interdir la lecture ou l'écriture du fichier au non root :

```
chmod 600 /etc/lilo.conf
```

LILO est maintenant un peu plus sécurisé.

Le fichier /etc/inittab.

Pour mémoire, inittab est le fichier de configuration responsable des services et fonctionnalités chargés lors du démarrage de Linux. Il existe deux modifications simples qui peuvent être apportées au fichier /etc/inittab pour rendre les paramètres par défaut d'init un peu moins vulnérables. La première est la modification du niveau d'exécution par défaut du système est la seconde et la suppression de la combinaison Ctrl+Alt+Suppr pour réinitialiser le système.

Typiquement, le niveau d'exécution par défaut pour les distribution disposant d'une interface graphique d'installation (Mandrake, Red Hat...) est de 5. Ce qui signifie que le système démarre automatiquement l'interface graphique au démarrage de la machine et vous propose une invite de login X. Cette fonctionnalité pose des problème de sécurité (voir X11). Il est conseillé d'utiliser un niveau d'exécution par défaut en mode console de niveau 3. Cela ne vous empêchera pas de lancer ultérieurement X en tapant à l'invite de commande : startx. Pour changer le niveau d'exécution, éditez votre fichier /etc/inittab en root. Trouvez la ligne correspondante à celle ci-dessous (normalement la première hors commentaires) et remplacez le 5 par 3 :

```
# Default runlevel.
# id:5:initdefault:
id:3:initdefault:
```

Pour désactiver la combinaison des trois touches, il suffit de mettre la ligne contenant " crtlaltdel " en commentaire en plaçant un # au début de la ligne comme dans l'exemple ci-dessous :

```
# Trap CTRL-ALT-DELETE
# ca::ctrlaltdel:/sbin/shutdown -t3 -r now
```

Le fichier /etc/securetty

Une étape importante et souvent oubliée dans la sécurisation des postes de travail est le fichier /etc/securetty qui énumère les tty sur lesquels le login root peut arriver. Si vous utilisez Telnet par exemple pour l'administration de votre poste, il se peut que cela pose des problèmes. Il est conseillé de mettre en commentaire toutes les entrées de ce fichier et n'y laisser que un ou deux périphériques vc. Le format du fichier est simple et ne devrait pas vous poser de problème :

```
#tty1
#tty2
#tty3
#tty4
#tty5
#tty6
vc/1
vc/2
```

Ainsi toute les tentatives de connexions root distantes sur les tty seront rejettées. Ouf ;-). Maintenant que tout semble dans l'ordre, attaquons-nous au coeur de Linux : le système et les utilisateurs.

Le système et les utilisateurs

Dans ce chapitre, je ne m'attarderai pas à expliquer le fonctionnement de init, des services et leurs configuration. Je vais simplement vous indiquer comment vérifier les processus qui sont lancés au démarrage de votre machine et comment les stopper. Nous verrons ensuite comment appliquer une politique de gestion des mots de passe des utilisateurs.

Trouver et arrêter les services inutiles

Si vous avez bien suivi ce qui a été dit précédement, alors le niveau d'exécution par défaut de votre Linux devrait être de 3. C'est-à-dire en mode console, multi-utilisateur et avec le réseau activé. Pour neutraliser un service inutile, rien de plus simple. Nous allons déplacer son lien symbolique dans un répertoire temporaire. Généralement, les liens vers les scripts d'initialisation des services se trouvent dans le répertoire /etc/rc.d/. Par exemple, si vous souhaitez arrêter le serveur de police xfs du niveau d'exécution 5, il suffit de retirer le lien symbolique du répertoire qui y est lié :

```
[root@bdm ~]# cd /etc/rc.d/rc5.d
(se rendre au répertoire des scripts de niveau 5)
[root@bdm rc5.d]# mkdir temporaire
(créer un dossier nommé temporaire)
[root@bdm rc5.d]# mv S90xfs temporaire
(déplacer le script S90xfs dans le répertoire temporaire)
```

Le nom des fichiers est à adapter à votre système, mais le principe est le même. Dès lors, que vous relancerez Linux, celui-ci ne chargera pas le service pour le niveau d'exécution choisi. Pour réactiver le service, redéplacer le script vers son répertoire original :

```
[root@bdm ~]# cd /etc/rc.d/rc5.d/temporaire
[root@bdm temporaire]# mv S90xfs ../
```

Connaître tous les services utiles sur sa machine n'est pas une mince affaire et dans ce domaine pas de magie. Il faut tester et retester, encore et encore, jusqu'à avoir une configuration qui vous convienne avec les services souhaités.

Shadow est-il activé ?

Une bonne chose à faire est de vérifier que les mots de passe shadow /etc/shadow sont bien installés sur votre système. Normalement, toutes les distribution récentes l'incluent automatiquement. En effet, sans cette précaution, n'importe qui serait à même de récupérer votre fichier de mot de passe /etc/passwd et d'utiliser un cracker brute force (du type : John The Ripper), afin de récupérer vos logins et compte root. Shadow est un fichier lisible uniquement par root qui contient votre password de manière cryptée. Pour vérifier la présence de ce fichier sur votre Linux, tapez à l'invite de commande sous votre compte utilisateur (non root) :

```
[user@bdm ~]$ cat /etc/shadow
cat: /etc/shadow: Permission denied
```

Si vous obtenez un autre message que celui-ci ou qu'il n'y a pas de fichier, alors c'est que shadow n'est installé sur votre système ou pas correctement. Pour remédier à ce défaut de sécurité, installer le package shadow depuis le cd de votre distribution ou de son site FTP.

Politique de gestion des mots de passe utilisateurs

Si vous lisez The Hackademy Journal, vous aurez peut-être remarqué l'article sur " Comment choisir un bon password " paru dans le numéro THJ 02. Une bonne solution pour la sécurité est d'imposer aux utilisateurs un changement de mot de passe périodique afin de couper l'herbe sous le pied aux crackers (casser un mot de passe crypté prend du temps, donc si les changements ont lieu avant que le cracker ait eu le temps de casser le précédent, tout le travail fourni par celui-ci sera inutile). Pour automatiser cette tâche, il existe une commande très utile : chage. Pour en savoir plus : man chage. Nous vous donnons ci-dessous un exemple d'automatisation :

```
chage -m 0 -M 60 -d 0 -I 0 -E 0 -W 3 brotha
```

Avec cette commande, l'utilisateur brotha devra changer son mot de passe à la prochaine connexion sur le système. Celui-ci devra le changer également tous les 60 jours et sera avertit de l'expiration 3 jours avant que le changement soit obligatoire.

Sécurisez votre accès au réseau !

Dans le cadre de la sécurité d'un poste de travail GNU/Linux connecté à un réseau local ou Internet, c'est l'étape obligatoire. Vous possédez un connexion haut débit par ADSL ou câble ? Imaginez que votre machine est une voiture ! Si vous ne prennez pas les précautions nécessaires, c'est un peu comme si vous la laissiez bien garée devant chez vous tous les soirs, portières ouvertes et clés sur le contact... A bon entendeur.

Le demon Internet inetd, votre pire cauchemar

Dans un grand nombre de distribution (ex : Mandrake 9.x), le service inetd a été remplacé par xinetd. Néanmoins, certaines l'utilisent encore et ce point ne devrait pas être évité (ex : Debian). Sur les machines de type Unix, la plupart des services réseaux sont associés à un demon spécifique qui a pour mission de prendre en charge les requêtes de connexions distantes. A chaque service son demon associé. Par

exemple, les requêtes Telnet seront prises en charges par *in.telnetd*, les requêtes FTP par in.ftpd, etc.

Dès lors il convient de s'assurer que ces demandes de connexions soient bien gérées par le système. Le fichier de configuration de inetd : /etc/inetd.conf est assez simple à comprendre. Il sera lu par le demon à chaque fois que celui-ci est lancé ou redémarré. Pour désactiver un service, il suffit simplement de mettre un # au début de la ligne pour la passer en commentaire. Dans l'exemple ci-dessous, nous avons gardé la possibilité d'accepter les connexions FTP, en refusant les autres :

```
ftp      stream  tcp  nowait  root  /usr/sbin/tcpd in.ftpd -l -a
#telnet  stream  tcp  nowait  root  /usr/sbin/tcpd in.telnetd
#shell   stream  tcp  nowait  root  /usr/sbin/tcpd in.rshd
#talk    stream  tcp  nowait  root  /usr/sbin/tcpd in.talkd
```

Comme vous pourrez le constater votre fichier devrait être plus important que celui-ci. En cas de doute sur tel ou tel service, il est conseillé de placer la ligne en commentaire. Pour réactiver le service, il suffira d'enlever le # et de redémarrer le demon. Pour activer ces modifications sans relancer votre machine : killall -HUP inetd

Comment modifier les ports par défaut des services ?

Nous n'allons pas revoir ici les principes de bases du fonctionnement du réseau. Sachez simplement pour mémoire qu'il existe pour chaque service réseau un port qui lui est associé. Il existe sur Linux un fichier de configuration qui se charge de faire cette relation : /etc/services. Voici un extrait de ce fichier :

```
ftp-data    20/tcp
ftp          21/udp
ssh          22/tcp
ssh          22/udp
telnet       23/tcp
smtp         25/tcp
time         37/tcp
```

Pour changer le port par défaut en écoute (port par défaut qui attend la connexion), il suffit de modifier l'information directement dans le fichier /etc/services. Par exemple, si on souhaite activer le service telnet pour une administration distante sur le port 9356 :

```
ftp-data    20/tcp
ftp          21/udp
ssh          22/tcp
ssh          22/udp
telnet       9356/tcp
smtp         25/tcp
time         37/tcp
```

C'est tout. Veillez à spécifier des numéros de port qui soient relativement élevé en gardant à l'esprit que les 1024 premiers sont réservés et que le dernier est 65535. Désormais pour se connecter par telnet sur la machine, il faudra préciser explicitement le port à contacter, soit :

```
telnet nom_ou_ip_de_la_machine 9356
```

L'arme fatale : TCP wrappers

TCP wappers est inclus depuis pas mal de temps dans les distributions Linux et fournit un niveau de sécurité essentiel pour les services gérés par inetd et pour le serveur ssh. Comme vous avez pu le voir dans l'exemple de fichier inetd ci-dessus, le demon appelé pour la plupart des services n'est pas le demon correspondant, mais /usr.sbin/tcpd le_nom_du_demon. Cela signifie qu'avant d'autoriser le demon à prendre en charge la connexion, celle-ci sera filtrée par le demon de TCP wappers : tcpd. Il existe principalement deux fichiers de configuration qui gèrent TCP wappers : /etc/hosts.deny qui se charge du refus de connexion et /etc/hosts.allow qui les autorise. Si vous souhaitez mettre en place une sécurité forte sur votre machine et que vous n'avez pas besoin de fournir des services à des machines distantes, alors il existe un moyen simple de refuser toutes les demandes de connexions. En root, éditez le fichier /etc/hosts.deny et placez l'unique ligne suivante :

```
ALL : ALL
```

Sachez aussi que le fichier /ctc/hosts.allow ne devrait rien contenir. Les règles fixées dans celui-ci étant prioritaire à celles de /etc/hosts.deny. Pour vérifier la validité des règles mises en place, il existe une commande simple : tcpdchk.

Une bonne idée : le firewalling avec ipchains/iptables

En matière de firewalling et de filtrage de paquets, ipchains (noyau 2.2) et aujourd'hui iptables (noyau 2.4) fournissent un moyen efficace pour mettre en place une sécurité forte sur un réseau ou sur un hôte. Néanmoins, pour des personnes ne maîtrisant pas parfaitement le fonctionnement de TCP/IP et de ses différents protocoles, ils peuvent sembler un peu durs à aborder. Pour vous en convaincre, je vous laisser consulter la page man : man iptables. Heureusement pour nous, le monde est bien fait et dans les domaines de l'open source rien n'est impossible. Je vous propose donc d'installer GuardDog (1) qui se chargera à votre place d'éditer et d'enregistrer vos règles pour ipchains/iptables suivant la version de votre kernel. Sachez tout de même que celui-ci ne fonctionne qu'avec KDE 2 et un noyau 2.2 mais que KDE 3 et un noyau 2.4 sont vivement recommandés. La configuration est très simple et ne devrait pas vous poser de difficultés particulières.

Une fois installé, vous devez lancer le programme en root et effectuer la première configuration. Par défaut, la zone Internet (c'est-à-dire le monde extérieur) et la zone locale (votre machine) sont déjà configurées et l'ensemble des règles en font un système complètement fermé. C'est donc à vous d'ouvrir les portes de votre machine au monde extérieur. Il vous suffit de sélectionner les services que vous souhaitez laisser entrer sur votre machine, comme HTTP, HTTPS, FTP ou POP3 et sélectionner les services qui doivent se connecter vers l'Internet comme SMTP par exemple.

X Window, votre meilleur ennemi

Une fois que vous aurez mis en place toutes ces modifications, normalement tous les ports de votre machine devraient être fermés sauf peut-être un : le port 6000. Celui-ci correspond à X Window qui fonctionne comme un serveur et permet les connexions distantes. Pour vérifier ce point, vous pouvez lancer nmap (2) sur votre machine et tester les ports ouverts :

```
[brotha@brotha etc]$ nmap -p 1-65535 localhost

Starting nmap V. 3.00 ( www.insecure.org/nmap/ )
Interesting ports on bdm (192.168.1.2):
(The 65530 ports scanned but not shown below are in state: closed)
Port       State       Service
6000/tcp   open        X11

Nmap run completed -- 1 IP address (1 host up) scanned in 4 seconds
```

Il apparaît donc clairement que X attend les demandes de connexion sur le port 6000. Il faut y remédier, surtout qu'il existe des exploits public de déni de sevice sur celui-ci. Pour le fermer définitivement, il suffit d'éditer en root le fichier /usr/X11R6/bin/startx et d'ajouter l'argument -nolisten TCP dans la ligne serverargs :

```
serverargs="-nolisten TCP"
```

Conclusion

Toutes les techniques mentionnées dans cette partie sont assez connues et s'adressent en priorité aux personnes ayant installé une distribution Linux dans le but de tester et découvrir le monde du libre, pour des stations de travail ne fournissant aucun service sur le réseau. Ce qui représente la majorité des cas pour des postes bureautiques et/ou multimédia connectés à Internet chez les particuliers. Ces mesures ne représentent en aucun cas une solution de sécurisation complète, mais peuvent être le point de départ pour une utilisation un peu plus secure de sa machine, en évitant de la laisser ouverte (trop facilement) aux nombreux scripts-kiddies qui peuplent aujourd'hui le réseau.

Ressources

Sécurité sous Linux – Editions CampusPress
(1) GuardDog : http://www.simonzone.com/software/guarddog/
(2) Nmap : http://www.insecure.org/nmap/

II. Crypter fichiers et e-mails avec GPG

Dans le monde de l'informatique, il existe aujourd'hui 3 manières de garder un document confidentiel : jeter le disque dur dans la Seine, l'encrypter, ou ne pas faire d'informatique. Il n'existe pareillement que 2 manières d'être sûr de l'identité d'un correspondant : le rencontrer directement, prendre ses empreintes digitales, vocales et rétiniennes (et encore, j'ai toujours des doutes), ou prendre ses empreintes digitales électroniques, à savoir sa signature numérique.

Le logiciel GNUpg est la version libre du célèbre Pretty Good Privacy, alias PGP. Il s'agit d'un soft permettant à la fois de signer numériquement un fichier, et / ou de l'encrypter, ainsi que de faire toutes les opérations utiles autour de ces deux activités.

Principe

GNUpg permet à la fois d'encrypter des mails, et de les signer. Il fonctionne sur le principe des clés asymétriques. Une clé privée est détenue uniquement par le possesseur de la clé, tandis qu'une clé publique est à la disposition de tout le monde.

Prenons un exemple. Jacques désire envoyer un mail confidentiel à Jean. Pour cela, il va encrypter son mail en utilisant la clé publique de Jean. A réception du mail, Jean va utiliser sa clé privée ainsi que la phrase de pass qui lui est associée pour décrypter le message de Jacques. Il va ensuite répondre à Jacques en utilisant la clé publique de celui-ci. Sur demande de Paul, Jean va ensuite devoir entrer en contact avec Timotee. Comme il s'agit d'un message de la plus haute importance, Jean va signer numériquement son message de telle sorte que Timotee soit bien sûr que c'est Jean qui lui ait écrit.

Mais tout ça pose un problème: comment Timotee peut-il être sûr que le mail vient bien de Jean? Il existe plusieurs manières de le savoir. Jean peut avoir déposé sa clé auprès d'une société comme VeriSign(tm), dont le rôle est justement de garder les clés des gens et de certifier (moyennant finances) que la clé leur appartient bien. Il faut pour cela une société dont la notoriété est suffisante pour que les deux parties en présence puissent lui faire confiance. On appelle d'ailleurs une telle société un "tiers de confiance". Seul problème: ils n'ont pas assez d'argent pour se payer les services de VeriSign. Si Jean et Timotee s'étaient déjà rencontrés, ils auraient pu échanger leur clé de mano a mano. Mais ce n'est pas le cas. Il leur reste une solution: le serveur de clés.

Paul et Timotee se sont déjà rencontrés plusieurs fois, et ont échangé leurs clés respectives. A cette occasion, ils ont signé mutuellement leurs clés. C'est-à-dire qu'une petite partie de la clé de Paul a été mise sur la clé de Timotee et vice versa. Les deux amis ont ensuite uploadé leurs clés mises à jour sur un serveur de clés du projet openpgp (http://pgp.mit.edu par exemple). Paul et Jean ont aussi échangé et signé leurs clés respectives. Le jour où Timotee reçoit un mail de Jean, il va chercher sa clé sur le serveur de clés, et là, il constate que son vieil ami Paul a signé sa clé. Il en conclut donc que la signature de Jean est relativement fiable, et que le Jean qui lui écrit est bien le même que celui de la signature. Voilà pour le principe général, nous allons maintenant passer à la pratique.

Pratique

Installation.

Les sources de gnupg sont sur :

ftp://ftp.gnupg.org/GnuPG/gnupg/gnupg-1.2.1.tar.gz.

L'installation se fait très rapidement:

```
(root@toto:/usr/src) tar xvzf gnupg-1.2.1.tar.gz
(root@toto:/usr/src) cd gnupg-1.2.1
(root@toto:/usr/src/gnupg-1.2.1) ./configure
```

```
(root@toto:/usr/src/gnupg-1.2.1) make
(root@toto:/usr/src/gnupg-1.2.1) make install
```

INITIALISATION.

Dans un premier temps, il nous faut générer un jeu de clés. Gnupg va nous générer une clé publique, une clé privée, un trousseau de clés (sur lequel on va mettre les clés téléchargées sur un serveur de clés), et un fichier de configuration (auquel nous ne toucherons pas).

```
(toto@toto:~) gpg --gen-key
gpg (GnuPG) 1.2.1; Copyright (C) 2002 Free Software Foundation, Inc.
This program comes with ABSOLUTELY NO WARRANTY.
This is free software, and you are welcome to redistribute it
under certain conditions. See the file COPYING for details.
Please select what kind of key you want:
   (1) DSA and ElGamal (default)
   (2) DSA (sign only)
   (5) RSA (sign only)
```

Ici, on choisit l'option par défaut puisqu'on veut avoir la possibilité de signer et d'encrypter ses mails.

```
DSA keypair will have 1024 bits.
About to generate a new ELG-E keypair.
              minimum keysize is  768 bits
              default keysize is 1024 bits
    highest suggested keysize is 2048 bits
What keysize do you want? (1024)
```

On va maintenant nous demander la taille de la clé d'encryptage. J'ai pour habitude de générer des clés de 4096 octets, car 2048 est désormais insuffisant. Le procédé est nettement plus long, et d'ailleurs le programme nous en avertit. Cela dit, le jeu en vaut la chandelle.

```
Please specify how long the key should be valid.
         0 = key does not expire
      <n>  = key expires in n days
      <n>w = key expires in n weeks
      <n>m = key expires in n months
      <n>y = key expires in n years
```

Il est recommandé de faire des clés d'un an maximum. Mais, de toute manière, mettez toujours une date limite d'utilisation, quitte à la changer plus tard dans le temps. Le soft nous demande maintenant plusieurs informations dont notre nom. Il est impératif de mettre sa véritable identité, car les échanges et signatures de clés se font systématiquement avec présentation d'une pièce d'identité. L'adresse e-mail doit être une adresse e-mail valide à laquelle la clé correspondra. Nous verrons par la suite comment ajouter d'autres adresses à une même clé. Le commentaire, enfin, est facultatif. Cela dit, il peut être utile pour indiquer qui on est en peu de phrases ("développeur" par exemple est succint, précis...). Un "OK" valide tout ça.

L'étape suivante est le choix de la passphrase: longue et compliquée, mais suffisamment simple pour s'en rappeler. Il s'agit d'une phrase et non d'un mot de passe, c'est-à-dire qu'une vingtaine de caractères est un minimum. Une fois la clé générée, nous vérifions avec:

```
(toto@toto:~) gpg --list-key
/home/toto/.gnupg/pubring.gpg
-----------------------------
pub   1024D/EB9F3A35 2002-12-10 toto toto (toto) <toto@toto.com>
sub   2048g/A5728E7D 2002-12-10 [expires: 2003-12-10]
```

Exportation des clés

Voila, nous pouvons maintenant signer nos messages, et décrypter les messages que l'on nous envoie. Mais tout cela ne sert a rien si personne ne peut récupérer notre clé publique. C'est pourquoi nous devons la mettre sur un serveur de clés. Une liste complète des ser-

veurs de clés du programme openpgp se trouve sur le site de gnupg :
http://www.gnupg.org.
On commence par extraire une version ascii de sa clé publique:

```
(toto@toto:~) gpg --export -a > toto.asc
```

On va ensuite se connecter avec un navigateur quelconque (lynx fait très bien l'affaire) sur le site http://pgp.mit.edu. Là, on copie / colle le contenu du fichier toto.asc dans le cadre prévu à cet effet, puis on soumet la clé. Dans moins de 24 heures, les paranoïaques du monde entier pourront télécharger votre clé publique et ainsi vous écrire des mails sécurisés.

Importation des clés

Dès lors que l'on reçoit un message encrypté ou signé, il faut être à même de pouvoir le décrypter ou en vérifier la signature. Pour cela, on se rend à nouveau sur le serveur pgp.mit.edu, et là, on rentre le mail de notre correspondant et on lance la recherche. On devrait obtenir en réponse une suite de 8 chiffres et lettres qui sont l'ID (c'est-à-dire l'identificateur) de notre correspondant. On lance alors:

```
(toto@toto:~) gpg --recv-key --keyserver pgp.mit.edu E4893BC2
```

Pour ajouter cette nouvelle clé à notre trousseau. Nous sommes maintenant prêts à vérifier l'authenticité des mails reçus. La commande gpg --list-key nous affiche désormais les clés récupérées en plus de la notre.

Signature de clés

Lors d'un de ses nombreux voyages, Paul a fait la connaissance de Philippe. Tous les deux paranos à souhait, ils ont décidé d'échanger leurs clés. Paul a inscrit sur un papier son nom, son mail et son empreinte digitale pgp (on parle de fingerprint). Il remet ensuite ce papier à Philippe accompagné de son passeport (ou toute autre pièce d'identité officielle), qui contrôle que les informations concordent. La récupération des fingerprint se fait comme cela:

```
(toto@toto:~) gpg --fingerprint > finger.asc
```

A ce sujet, je vous recommande de vous faire des cartes de visite contenant toutes ces informations, c'est toujours plus pratique que de les écrire sur un vieux bout de papier.

Une fois l'échange fait, il ne reste plus à Paul qu'à signer la clé de Philippe.

```
(toto@toto:~) gpg --recv-key AB12CD34
(toto@toto:~) pgp --sign-key AB12CD34
```

A ce moment là, il nous est demandé notre phrase de pass. Normal puisque nous allons exporter un petit bout de notre clé privée sur la clé que nous signons pour indiquer que nous l'avons bien signée. Un examen de la clé publique (par exemple via le serveur pgp.mit.edu) permet de voir tous les signataires.

Il ne reste plus à Paul qu'à exporter la clé nouvellement signée sur le serveur où il l'a prise, en ayant auparavant mis cette clé à jour:

```
(toto@toto:~) gpg --recv-key --keyserver pgp.mit.edu AB12CD34
(toto@toto:~) gpg --send-key --keyserver pgp.mit.edu AB12CD34
```

Lors d'un échange de clés exigez systématiquement une pièce d'identité valide de la part de votre contact (qui devrait faire de même pour vous). Il se peut que votre correspondant ait sur une même clé plusieurs ID avec plusieurs mails et qu'il n'ait pas mentionné ces mails durant l'échange. Dans ce cas, ne signez que les mails qu'il vous a mentionné. Et refusez systématiquement de signer la clé de quelqu'un qui refuse de vous présenter ses papiers d'identité. En effet, après votre signature, vous vous portez garant de l'authenticité de sa clé. C'est ce qu'on appelle une relation de tiers de confiance. Il est ensuite possible de mettre une note de confiance à chacun de ses correspondants: elle sera en général fonction du nombre de personnes de notre trousseau en qui nous avons confiance et qui ont signé la clé de cette personne, ou le contraire.

Encryptage et décryptage

N'oublions pas qu'une des principales missions de gnupg est l'encryptage de documents ou de fichiers exécutables. Je vous rappelle le principe: on va encrypter le fichier avec la clé publique de notre correspondant, de telle sorte qu'il puisse le décrypter avec le couple clé privée + passphrase.

Paul désire envoyer un message ultra secret à Philippe. Une fois ce message entré, il va l'encrypter avec la clé publique de Philippe :

```
(toto@toto:~) gpg --encrypt AB12CD34 message.txt
```

Offre spéciale d'abonnement

LINUXSCHOOL
Magazine

LE NOUVEAU MAGAZINE 100% LINUX

1 an

de pur linux (6numéros)

24 euros !!!

A découper ou à recopier et à envoyer accompagné de votre réglement de 24 euros à l'ordre de LPN à LINUXSCHOOL Magazine15 rue Chevreul - 94700 MAISONS-ALFORT

NOM : _____ PRENOM : _____

ADRESSE : _____

CODE POSTAL : _____ VILLE : _____

ou encore:

```
(toto@toto:~) gpg -e AB12CD34 message.txt
```

Lorsque Philippe reçoit le message, il n'a plus qu'à le décrypter avec sa clé privée:

```
(philippe@tata:~) gpg --decrypt message.txt
```

ou encore

```
(philippe@tata:~) gpg -d message.txt
```

GNUpg lui demande alors sa passphrase, puis le message s'affiche en clair.

Gestion des utilisateurs

Il nous est bien entendu possible de gérer notre trousseau de clé, et notamment rajouter de nouveaux identificateurs, par exemple si j'ouvre une nouvelle boite mail, ou encore un ID professionnel et un ID personnel.

On commence par entrer en mode édition :

```
(toto@toto:~) gpg --edit-key EB9F3A35
```

On va ajouter un utilisateur à notre clé. Mais attention: cet utilisateur n'apparaîtra pas comme ayant été signé par les clés ayant déjà signé notre clé, ce qui est bien normal.

```
Command> adduid
Real name: Toto toto
Email address: toto@totomail.com
Comment: clé personnelle
You selected this USER-ID:
    "Toto toto (clé personnelle) <toto@totomail.com>"
```

Il nous est ensuite demandé de rentrer notre passphrase pour valider ce changement. Une fois sorti du mode édition, gpg --list-key nous affiche:

```
(1)  toto toto (toto) <toto@toto.com>
(2). Toto toto (clé personnelle) <toto@totomail.com>
```

Après toute modification de votre clé, n'oubliez pas de la mettre à jour sur votre serveur de clés préféré, histoire que vos correspondants puissent eux aussi profiter de ces modifications.

Signer des documents et vérifier des signatures

Pour signer un document, lançons la commande suivante :

```
(toto@toto:~) gpg --sign document.txt
```

ou encore

```
(toto@toto:~) gpg -s document.txt
```

Il nous est alors demandé notre passphrase, puis le fichier signé sort au format .asc.

Si nous désirons maintenant vérifier la signature apposée sur un document, il suffit (une fois la clé recupérée sur un serveur de clés si ce n'était pas déjà le cas) de taper les commandes suivantes :

```
(toto@toto:~) gpg --verify document.asc
```

Utilisation quotidienne

Pour conclure, rappelons que la plupart des clients mail modernes supportent complètement GNUpg. C'est par exemple le cas de kmail, le mailer de KDE. Mutt, à mon avis le meilleur logiciel de mail qui soit, le supporte aussi, à condition d'ajouter quelques lignes dans le fichier de configuration .muttrc.

```
# Pour décrypter un mail
set pgp_decode_command="/usr/bin/gpg   %?p?--passphrase-fd 0? --no-verbose --quiet  --batch  --output - %f"
# Nous vérifions l'authenticité d'une signature
```

```
set pgp_verify_command="/usr/bin/gpg    --no-verbose --quiet  --batch  --output --verify %s %f"

# Pour décrypter une pièce jointe
set pgp_decrypt_command="/usr/bin/gpg    --passphrase-fd 0 --no-verbose --quiet  --batch  --output - %f"

# Pour signer un document en pièce jointe.
set pgp_sign_command="/usr/bin/gpg     --no-verbose --batch --quiet    --output - --passphrase-fd 0
--armor --detach-sign --textmode %?a?-u %a? %f"

# Pour signer un message avec GNUpg
set pgp_clearsign_command="/usr/bin/gpg    --no-verbose --batch --quiet    --output - --passphrase-
fd 0 --armor --textmode --clearsign %?a?-u %a? %f"

# Pour encrypter une pièce jointe
set pgp_encrypt_only_command="pgpewrap /usr/bin/gpg     --batch  --quiet  --no-verbose --output -
--encrypt --textmode --armor --always-trust -- -r %r -- %f"

# Pour encrypter et signer une pièce jointe.
set pgp_encrypt_sign_command="pgpewrap /usr/bin/gpg  --passphrase-fd 0  --batch --quiet  --no-ver-
bose  --textmode --output - --encrypt --sign %?a?-u %a? --armor --always-trust -- -r %r -- %f"

# Pour importer une clé dans notre trousseau de clés publiques.
set pgp_import_command="/usr/bin/gpg --no-verbose --import -v %f"

# Pour exporter cette même clé
set pgp_export_command="/usr/bin/gpg    --no-verbose --export --armor %r"

# Pour vérifier l'authenticité d'une clé
set pgp_verify_key_command="/usr/bin/gpg    --verbose --batch  --fingerprint --check-sigs %r"

# Pour lire dans le trousseau de clés publique
set pgp_list_pubring_command="/usr/bin/gpg    --no-verbose --batch --quiet    --with-colons --list-
keys %r"

# Pour lire dans le trousseau de clés secrètes
set pgp_list_secring_command="/usr/bin/gpg    --no-verbose --batch --quiet    --with-colons --list-
secret-keys %r"
```

III. Firewaller Un système Linux

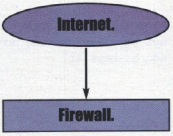
Un firewall est souvent considéré, à tort, comme le rempart ultime contre les pirates. C'est faux, et les gens qui vous vendent un firewall en vous disant qu'avec lui, vous serez protégés contre tout et n'importe quoi sont exactement comme ceux qui vous promettent que la dernière version d'un antivirus vous protège contre toute menace ; des charlatans. Cela dit, il est exact qu'un firewall peut parfois s'avérer utile pour se protéger, autant des agressions extérieures qu'intérieures. Eh oui, rappelez-vous : il n'existe pas de gentils utilisateurs, il n'existe que des désastres potentiels...

Nous allons nous intéresser ici aux firewalls sous linux, et, plus particulièrement, aux firewalls inclus dans le kernel. En effet, ils ne présentent que des avantages, et aucun inconvénient : ils sont inclus au plus bas niveau du système; leurs sources sont ouvertes, ce qui permet un contrôle au niveau sécuritaire. Ainsi, on est sûr que l'outil que l'on a choisi pour se protéger n'est pas celui qui va servir au pirate à nous rooter. On a ainsi vu, sous Windows, des firewalls backdoorés, voir même des portscanners backdoores, comme le célèbre 7th sphere portscan.

S'il n'existe pas, à priori, de bonne méthode de firewalling, il en existe en revanche des catastrophiques. Nous allons tâcher de les éviter autant que possible, tout en vous montrant les "bons réflexes" à avoir.

Dans un premier temps, nous allons nous pencher sur les différentes architectures de firewall, avant de voir l'implémentation au niveau des kernels 2.0, 2.2 et 2.4.

Cet article ne comporte à priori qu'un seul pré-requis: être root sur une box tournant sur un nunux pas trop trop vieux (c'est a dire postérieur à 1998), et encore, c'est juste pour les travaux pratiques.



Architectures firewall

Dans un premier temps, il va nous falloir sélectionner une architecture de firewalling. Celle-ci sera le pilier de toute l'architecture de notre réseau local. Tout dépendra bien évidemment des moyens a notre disposition et de la taille du réseau que nous avons à notre disposition. Les exemples que je vais vous proposer ici vous seront présentés dans l'ordre d'efficacité.

Firewall basic :

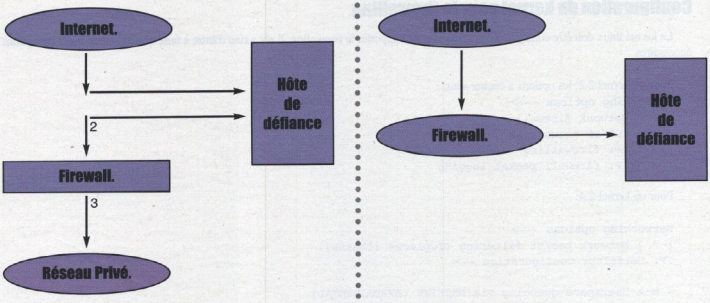
Comme montré sur la figure ci-contre, ce type de firewall est la base de tout firewaling. Une simple machine faisant office de firewall fait barrage entre un réseau non sûr, en général le net, et un réseau local. C'est une solution peu coûteuse dans laquelle la machine pare-feu effectue toutes les fonctions de firewalling.

L'hôte de défiance :

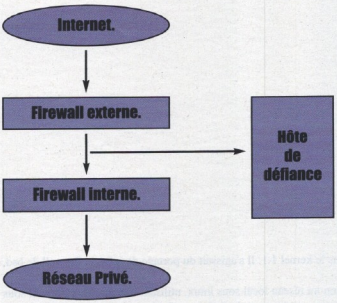
Il s'agit en fait d'une machine située entre le réseau non sûr et le firewall, comme montre la page suivante. Le firewall est configuré de telle manière que toutes les connections, tant entrantes que sortantes, doivent passer par lui. Il est appelé hôte de défiance, car il est placé sur le réseau non sûr en deçà du firewall. Celui-ci ne peut donc pas le protéger. Il s'agira toujours d'une machine configurée à la fois le plus légèrement et de la manière la plus secure. Bien évidemment, on ne doit placer qu'une confiance très limitée dans les hôtes protégés par le firewall. Cela dit, en tant qu'administrateur système, j'ai rayé les termes comme confiance et tous ses synonymes de mon vocabulaire.

La zone démilitarisée (ou DMZ pour faire élite)

Cette configuration comporte elle aussi un hôte de défiance, mis à part qu'il est placé à l'intérieur même du firewall. Simplement, il



est sur son propre sous réseau, ce qui fait que le firewall comporte trois cartes réseau. Bien entendu, il n'est pas plus question d'avoir confiance dans les hôtes se trouvant derrière le firewall, Cette configuration est parfaite pour placer un ftp public ou encore un serveur web tout en gardant notre réseau local à l'abris.



FIREWALL DOUBL

Dans cette configuration, le réseau local est isolé de l'hôte de défiance par un second firewall, ou firewall interne. Les paquets devront ici traverser les deux firewalls et l'hôte de défiance avant de passer du réseau local au net et vice versa.

Dans ces exemples, nous avons étudié une situation classique, à savoir l'utilisation d'un firewall pour protéger un réseau privé d'un réseau public. Mais ces implémentations peuvent aussi servir à protéger divers sous réseaux d'un même réseau local les uns des autres, dans le cas d'un réseau d'entreprise ou d'université.

Implémentations et exemples

Dans cette seconde partie, nous allons nous pencher tout particulièrement sur les firewalls contenus dans les kernel linux, version 2.0, 2.2 et 2.4. Le kerneel 2.5, actuellement en cours de développement propose lui aussi un nouveau type de firewall, mais son développement n'est pas assez avancé pour qu'il soit possible d'écrire quoi que ce soit dessus sans que cela soit obsolète avant même la parution du papier.

Pour configurer un firewall sous linux, il faut, dans un premier temps, installer le support du firewalling dans le noyau.

Les kernels antérieurs à la version 2.2 utilisent l'utilitaire ipfwadm; les kernels de la génération 2.2 utilisent ipchains. A partir de la version 2.3.15 est apparu netfilter, aujourd'hui utilisé dans les kernel de la famille des 2.4. Ce dernier utilitaire a vu le " complete redefinition " de la gestion des paquets au sein du kernel linux, et son utilisation est des plus intéressantes. Il supporte la syntaxe ipchains et ipfwadm, ce qui permet a chacun de garder la syntaxe qui lui plaît le plus, ou avec laquelle il se sent le plus à l'aise.

Il existe deux types de politique de firewalling: la première, et la pire, consiste à tout autoriser et à n'interdire que ce que l'on veut interdire. C'est une politique de feignasse; elle est donc catastrophique. La seconde consiste à tout bloquer et à ne laisser passer que les paquets autorisés. Ce sera notre politique par défaut.

Configuration du kernel pour le firewalling

Le kernel linux doit être configuré correctement pour supporter le firewaling. Il n'y a rien d'autre à faire que de sélectionner les options nécessaires.

Pour un kernel 2.2, les options a cocher sont:

```
Networking options --->
[ * ] Network firewalls
[ * ] TCP/IP networking
[ * ] IP: firewalling
[ * ] IP: firewall packet logging
```

Pour un kernel 2.4:

```
Networking options --->
[ * ] Network packet filtering (replaces ichains).
IP: Netfilter configuration -->
    .
< M > Userspace queueing via NETLINK (EXPERIMENTAL)
< M > IP tables support (required for filtering/masq/NAT)
< M > Limit match support
< M > MAC adress match support
< M >  Netfilter MARK match support
< M >  TOS match support
< M >  Connection state match support
< M > Unclean Match Support (EXPERIMENTAL)
< M > Owner match support (EXPERIMENTAL)
< M > Packet filtering
< M > REJECT target support
< M > MIROR target support
    .
< M >  Packet mangling
< M > TOS target support
< M > MARK target support
< M > LOG target support
< M > ipchains (2.2 style) support
< M > ipfwadm (2.0 style) support
```

Utilisation d'ipfwadm

La première implémentation de firewalls sous linux se trouvait dans le kernel 1.1. Il s'agissait du portage de ipfw, le firewall de bsd, par Alan Cox.

Ici, nous allons partir du principe que nous avons à notre disposition un réseau local sous linux, utilisant une machine firewall sous linux pour se connecter au net. Nous désirons que nos utilisateurs puissent accéder au web, mais tout en bloquant tout autre trafic. Nous allons créer une règle de forwarding pour permettre aux utilisateurs d'émettre des données via la port 80, et d'en recevoir la réponse.

Notre réseau est un réseau de class C, et son adresse est 171.15.17.0.

```
# ipfwadm -F -f
# ipfwadm -F -P deny
# ipfwadm -F -a accept -P tcp -S 171.15.17.0/24 -D 0/0 80
# ipfwadm -F -a accept -P tcp -S 0/0 80 -D  171.15.17.0/24
```

L'option -F indique qu'il s'agit d'une règle de forwarding.

La première ligne indique à ipfwadm de vider sa mémoire de toutes les règles précédemment établies. Cela nous permet de travailler en terrain connu.

La seconde ligne indique notre politique par défaut. Elle est très importante car elle va définir ce qui va arriver à tous les paquets qui ne matchent pas les règles d'exception. Ici, bien entendu, nous posons une politique de deny sur tout.

Les deux lignes suivantes sont celles qui implémentent notre politique. Il existe une option, -b, qui permet a une règle de devenir bidirectionnelle. Nos deux lignes deviendraient alors:

```
# ipfwadm -F -a accept -P tcp -S  171.15.17.0/24  -D 0/0 80 -b
```

- **F** indique une règle de fowarding.
- **A ACCEPT**: indique que les paquets correspondant à cette règle sont acceptés.
- **P TCP** indique que cette règle vaut pour le protocole tcp, à l'opposé d'UDP ou d'ICMP.
- **S 171.15.17.0/24** indique que l'adresse d'émission doit avoir 24 bits en correspondance avec la règle. Ici, il s'agit de notre réseau.
- **D 0/0 80** signifie que l'adresse de destination peut être n'importe laquelle pourvu que le port soit 80.

Notre politique contient une grosse faille de sécu: n'importe qui peut se connecter au port 80 depuis l'extérieur et ainsi nous balancer un gros SYN flooding. Nous allons corriger ça avec la chaîne :

```
# ipfwad -F -a deny -P tcp -S 0/0 80 -D  171.15.17.0/24 -y
```

Pour lister les règles actuellement en cours :

```
# ipfwadm -F -l
```

Utilisation d'ipchains

La création d'ipchains a découlé de la demande de plus en plus forte des utilisateurs d'un outil plus simple et plus convivial qu'ipfwadm. De plus, ce dernier pouvait se révéler totalement inefficace dans certaines situations délicates. Ipchains a ainsi introduit une nouvelle syntaxe plus claire :

-A chaine :
Ajoute une règle à la fin de toutes celles existant déjà. Si jamais un hostname est donné et qu'il correspond à plusieurs adresses IP, alors, une règle sera créée pour chacune des adresses.

-I chaine :
Insère une règle au début de la chaîne. Une fois de plus, si un hôte correspond à plusieurs adresses IP, une règle sera ajoutée pour chacune des IP.

-D chaine :
Efface une règle de la chaîne actuelle si elle correspond aux spécifications demandées.

-D chaine, numéro de règle.
Cette option permet de retirer la règle spécifiée dans la chaîne. La spécification commence à un, une fois n'est pas coutume.

-R chaine, numéro de règle
Remplace la règle indiquée dans la chaîne par la nouvelle.

-L :
Permet de lister les règles d'une chaîne.

-F chaine:
Remet la chaîne spécifiée à zéro, où toutes les chaînes si aucun nom n'est indiqué.

-N chaine:
Crée une nouvelle chaîne avec le nom spécifié.

-P chaine politique
Met en place la politique par défaut de la chaîne spécifiée. Les politiques valides sont ACCEPT, DENY, REJECT, REDIR ou RETURN. Les politiques ACCEPT, DENY et REJECT ont la même spécification que sur n'importe quel utilitaire de firewaling. Redir indique que le datagramme doit être redirigé vers un autre port du firewall.

PARAMÈTRES DES RÈGLES D'IPCHAINS:
-p [!]protocole: spécifie le protocole à laquelle la règle va correspondre. Les protocoles valides sont tcp, udp et icmp, ainsi que all.

Si le '!' est utilisé, il indique que la règle ne prends pas ce protocole en compte.

-s [!]adresse/mask [!]port:

Spécifie l'adresse IP source du datagramme dans la règle à prendre en compte. Très utile pour empêcher un emmerdeur de revenir à la charge. L'adresse peut être un nom de réseau, un nom de domaine ou une adresse IP. Le port peut être soit un numéro de port, soit une intervalle. Par exemple 21:23 indique que les ports 21 a 23 compris seront pris en compte.

-d [!]adresse/mask [!]port:

Spécifie l'adresse de destination et le port du datagramme correspondant à la règle. La manière d'utiliser ce paramètre est la même que pour -s

-j cible:

Cette option indique quelles mesures prendre quand la règle correspond au datagramme reçu. Les cibles valides sont ACCEPT, DENY, REJECT, REDIR et REJECT.

-i [!] interface:

Spécifie l'interface sur laquelle s'applique la règle. Un + après le nom du device indique tous les devices de ce type. Par exemple eth+ indique toutes les cartes ethernet.

Ainsi, avec ipchains, notre exemple de tout à l'heure deviendra :

```
# ipchains -F forward
# ipchains -p forward DENY
# ipchains -A forward -s 0/0 80 -d  171.15.17.0/24 -p tcp -y -j DENY
# ipchains -A forward -s  171.15.17.0/24  -d 0/0 80 -p tcp -j ACCEPT
```

La première règle remet toutes les règles de forwarding à zéro.
La seconde indique DENY comme politique de forwarding par défaut.
La troisième indique DENY sur tous les paquets provenant du port 80 et ayant le bit SYN à 1.
La dernière, enfin, accepte le forwarding sur le port 80 en flux entrant autant qu'en flux sortant.

Utilisation d'iptables

Iptables est l'utilitaire de netfilter pour le firewalling. Il est extrêmement flexible; ainsi, il est totalement compatible avec les anciennes règles ipchains et ipfwadm. Ainsi, pas la peine de refaire ses scripts de 2000 lignes quand on change de kernel. Pour cela, il faut installer les modules ipfwadm.o et ipchains.o de la manière suivante :

```
# modprobre ipfwadm
# modprobe ipchains
```

Avant de pouvoir utiliser iptables, il faut d'abord charger le module correspondant en mémoire :

```
# modprobe ip_tables
```

La commande iptables est utilisée tant pour mettre en place le filtrage de paquets que pour le NAT (Network Adresse Translation). Pour clarifier les choses, il existe ainsi deux chaînes de règles, nommées filter et nat. Les chaînes INPUT et FORWARD sont utilisées par la table filter; les chaînes PREROUTING et POSTROUTING sont utilisées par la table NAT. Enfin. La règle OUTPUT est utilisée par les deux.

Je ne m'intéresserai ici qu'à la table filter, correspondant au filtrage de paquets. En effet, le NAT n'est pas ma préoccupation dans cet article.

Les commandes d'iptables

-A chaine:

Ajoute une règle à la fin de la chaîne indiquée. Si un nom d'hôte est donné et qu'il correspond à plusieurs adresses ip, alors une règle est créée pour chacun des cas.

-I chaine numéro de règle:

Insère une ou plusieurs règles au début de la chaîne passée en paramètre. Une fois de plus, si vous ou vos collaborateurs étiez capturés ou tués, le département d'état.... etc etc etc.

-D chaine :

Supprime la règle correspondante de la chaînes.

-D chaine, numéro de règle.
Cette option permet de retirer la règle spécifiée dans la chaîne. La spécification commence à un, une fois n'est pas coutume.

-R chaine, numéro de règle
Remplace la règle indiquée dans la chaîne par la nouvelle.

-L:
Permet de lister les règles d'une chaîne.

-F chaine :
Remets la chaîne spécifiée à zéro, ou toutes les chaînes si aucun nom n'est indiqué.

-N chaine :
Crée une nouvelle chaîne avec le nom spécifié.

-P chaine politique
Mets en place la politique par défaut de la chaîne spécifiée. Les politiques valides sont ACCEPT, DROP, QUEUE, ou RETURN. Accept permet au datagramme de passer tandis que DROP le rejette tout simplement.

- PARAMÈTRES DES RÈGLES D'IPTABLES

-p [!]protocole: spécifie le protocole auquel la règle va correspondre. Les protocoles valides sont tcp, udp et icmp, ainsi que all. Si le '!' est utilisé, il indique que la règle ne prends pas ce protocole en compte.

-s [!]adresse/mask
Spécifie l'adresse IP source du datagramme dans la règle à prendre en compte. Très utile pour empêcher un emmerdeur de revenir à la charge. L'adresse peut être un nom de réseau, un nom de domaine ou une adresse IP.

-d [!]adresse/mask
Spécifie l'adresse de destination et le port du datagramme correspondant à la règle. La manière d'utiliser ce paramètre est la même que pour -s

-j cible :
Cette option indique quelles mesures prendre quand la règle correspond au datagramme reçu. Les cibles valides sont ACCEPT, DENY, REJECT, REDIR et REJECT.

-i [!] interface :
Spécifie l'interface sur laquelle s'applique la règle. Un + après le nom du device indique tous les devices de ce type. Par exemple eth+ indique toutes les cartes ethernet.

-o [!] interface :
Specifie l'interface à laquelle le paquet va être transmis. La syntaxe est la même que pour l'option -i.

Extensions TCP : elles s'utilisent avec les flags -m tcp et -p tcp.

--sport [!] port [port:port]:
indique le ou les ports source que doit utiliser le datagramme pour correspondre à la règle. Un '!' indique que le port ne doit pas être utilisé. Une tranche de ports peut aussi être donnée grâce aux ':': 21:23 indique que les ports 21 a 23 correspondent à la règle.

--dport [!] port [port:port]:
indique le ou les ports destination que doit utiliser le datagramme pour correspondre à la règle. Un '!' indique que le port ne doit pas être utilisé. Une tranche de ports peut aussi être donnée grâce aux ':': 21:23 indique que les ports 21 a 23 correspondent à la règle.

--tcp-flag [!] mask
Indique le flag du paquet correspondant à la règle indiquée: ils peuvent être SYN, ACK, FIN, RST, URG, PSH, ALL et NONE. Ce sont des options très avancées.

Extentions UDP : elles s'utilisent avec les flags -m udp -p udp.

--sport [!] port [port:port]:
indique le ou les ports source que doit utiliser le datagramme pour correspondre à la règle. Un '!' indique que le port ne doit pas être utilise. Une tranche de ports peut aussi être donnée grâce aux ':': 21:23 indique que les ports 21 a 23 correspondent à la règle.
--dport [!] port [port:port]:
indique le ou les ports destination que doit utiliser le datagramme pour correspondre à la règle. Un '!' indique que le port ne doit pas être utilise. Une tranche de ports peut aussi être donnée grâce aux ':': 21:23 indique que les ports 21 a 23 correspondent à la règle.

EXEMPLE CONCRET :
Une fois de plus, nous reprenons notre exemple utilise plus haut.

```
# modprobe ip_tables
# iptables -F forward
# iptables -P FORWARD DROP
# iptables -A FORWARD -m tcp -p tcp -s 0/0 --sport  80 -d  171.15.17.0/24 --syn -j DROP
# iptables -A FORWARD -m tcp -p tcp -s 171.15.17.0/24 --sport 80 -d 0/0 -j accept
# iptables -A FORWARD -m tcp -p tcp -d 171.15.17.0/24 --dport 80 -s 0/0 -j ACCEPT
```

Dans un premier temps, nous avons charge le module ip_tables en mémoire. A cet endroit, il me faut faire une petite digression: aujourd'hui, de très nombreux rootkit utilisent des modules kernel pour se dissimuler. Même si un système compromis réclame une réinstallation d'urgence, il vaut mieux, à mon avis, ne pas faciliter la tâche du pirate en compilant notre noyau avec le support des modules. Après tout, sur un firewall, on ne change pas le hardware si souvent que ça non ?
Notre seconde ligne remet à zéro toutes les règles de fowarding.
La troisième indique que la politique par défaut des règles de forwarding est le rejet.
Notre quatrième ligne indique que les paquets venant sur le port 80 et ayant le bit SYN às 1 seront rejetés.
Nos deux dernières lignes acceptent les paquets sur le port 80 en entrée comme en sortie.

Conclusion

Soyez toujours très prudents avec les règles de firewall car elles peuvent parfois favoriser le pirate alors que c'est exactement le contraire qu'on leur demande. Et souvenez vous bien : le seul serveur secure est celui que l'on débranche du réseau, dont on balance la prise au feu et le disque dur dans la seine. Have fun :-)

APPENDICE :
Un script iptables générique. Ce script correspond à la plupart des situations que vous pourrez rencontrer. Il est à lancer au démarrage. Sous Slack, rajoutez les lignes suivantes dans /etc/rc.d/rc.inet2, et copiez ce script dans /etc/rc.d/ rc.firewall. N'oubliez pas de changer les droits en exécution.

```
if [ -x /etc/rc.d/rc.firewall ]; then
/etc/rc.d/rc.firewall
echo "    Starting firewalling"
fi
>--- >--- >--- >--- >--- >--- >--- >--- >--- >--- >--- >--- >---
#!/bins/sh

# Section utilisateur: modifiez selon votre réseau et vos envies.

IPTABLES=iptables

PATH="/sbin"

MYNET="171.15.17.0/24"
MYBCAST="171.15.17.255"
MYIFACE="eth0"

ANYADDR="0/0"
```

```
ANYDEV="eth1"

TCPIN="smtp, pop3, www, ftp, ftp-data"
TCPOUT="smtp, pop3, www, ftp, ftp-data, ircs, nntp"

UDPIN="domain"
UDPOUT="domain"

ICMPIN=" "
ICMPOUT="8 3 11"
```

 # Fin de la section utilisateurs. On passe aux choses sérieuses.

 # Remise à zéro des règles.

```
$IPTABLES -F FORWARD
```

 # On bloque tout et même le reste.

```
$IPTABLES -P FORWARD deny
```

 # On bloque tous les paquets entrants.

```
$IPTABLES -A INPUT -i $ANYDEV -j DROP
```

 # Ca c'est pour empêcher l'IP spoofing

```
$IPTABLES -A FORWARD -s $MYNET -i $ANYDEV -j DROP
```

 # Et ca, c'est pour le smurf

```
$IPTABLES -A FORWARD -m multiport -p icmp -i $ANYDEV -d $MYNET -j DENY
```

 # On accepte les paquets fragmentés.

```
$IPTABLES -A FORWARD -f -j ACCEPT
```

 # On accepte tous les paquets TCP appartenant à une connexion existante.

```
$IPATBLES -a FORWARD -m multiport -p tcp -d $MYNET –dports $TCPIN ! --tcp-flag SYN< ACK ACK -j
ACCEPT
$IPATBLES -a FORWARD -m multiport -p tcp -d $MYNET –sports $TCPIN ! --tcp-flag SYN< ACK ACK -j
ACCEPT
```

 # Politique d'acceptation de paquets TCP: seulement les ports autorisés.

```
$IPTABLES -A FORWARD -m multiport -p tcp -i $ANYDEV -d $MYNET $TCPIN –syn -j ACCEPT
```

 # Politique des sorties de paquets TCP, juste sur les ports autorisés.

```
$IPTABLES  -A FORWARD -m multiport -p tcp -i $MYDEV -d $ANYADDR –dports $TCPOUT –syn -j ACCEPT
```

 # Politique d'acceptation des paquets UDP entrants.

```
$IPTABLES -A FORWARD -m multiport -p udp -i $ANYDEV -d $MYNET –dports $UDPIN -j ACCEPT
$IPTABLES -A FORWARD -m multiport -p udp -i $ANYDEV -s $MYNET –dports $UDPIN -j ACCEPT
```

 # Sortie des paquets UDP.

```
$IPTABLES -A FORWARD -m multiport -p udp -i $MYDEV -d $ANYADDR –dports $UDPOUT -j ACCEPT
$IPTABLES -A FORWARD -m multiport -p udp -i $MYDEV -s $ANYADDR –dports $UDPOUT -j ACCEPT
```

 # Paquets ICMP entrants: on ne répond pas aux pings

```
$IPTABLES -A FORWARD -m multiport -p icmp -i $ANYDEV -d $MYNET –dports $ICMPIN -j DROP
```

 # Paquets ICMP sortant: juste ce qu'il faut.

```
$IPTABLES -A FORWARD -m multiport -p icmp -i $MYDEV -d $ANYADDR –dports %ICMPOUT -j ACCEPT
```

 # C'est fini pour cette fois !

```
>--- >--- >--- >--- >--- >--- >--- >--- >--- >--- >--- >--- >---
```

IV. Sécuriser son Linux avec les patches kernel de GRSecurity

Renforcez la sécurité du noyau de votre linbox ! Nous vous proposons de découvrir ici un des meilleurs outils, complet et open-source, pour garantir au mieux l'intégrité globale de votre machine.

Note importante : Pour comprendre cette dernière partie, il vous faut savoir comment recompiler votre noyau Linux. Pour des raisons de place, nous n'avons pu détailler ces notions ici. Toutefois, vous pouvez trouver des explications complètes sur la recompilation du kernel Linux dans le cours officiel de the Hackademy.

Le fait que les sources du noyau Linux soient libres a permis le développement de patches permettant de rajouter au noyau des fonctionnalités non prévues par la team de développement du kernel. Ainsi, avant d'être ajouté au kernel, le système de fichier reiserfs était-il fourni sous forme de patch. C'est aussi le cas pour des fonctionnalités de sécurité qu'on ne trouve pas dans le kernel de base et qui sont intéressantes dès lors que l'on va avoir une machine reliée en réseau et comptant plusieurs utilisateurs.

GRsecurity fait partie de ces patches. Téléchargez-le sur http://www.grsecurity.net, appliquez le patch sur les sources de votre noyau, puis lancez la configuration du noyau :

```
cd /usr/src
patch -p0 < grsecurity-1.9.12-2.4.22.patch
cd linux
make menuconfig
```

Il ne s'agit pas en fait d'un patch en soi, mais plutôt d'un regroupement de divers patches (un patchwork quoi), proposant les diverses protections que l'on peut attendre de ce genre d'outil. Dans un premier temps, nous constatons que GRsecurity est encore en phase de développement sur les kernels 2.4.x, ce qui signifie que l'utilisation se fait à vos risques et périls. Cependant, nous l'utilisons sur nos machines personelles et sur nos serveurs, sans aucun problème. GRsecurity nous propose 4 niveaux de sécurité différents: low, medium, high et cutomised. Il est déconseillé de choisir un des trois premiers pour une station de travail. En effet, certains éléments empêcheraient certaines applications (comme le serveur X) de se lancer. Nous allons donc nous plonger dans les entrailles du customized level.

Buffer overflow protection

Cette première section va nous permettre de nous prémunir contre les failles de type dépassement de tampons classiques. Les correctifs proposés ne les empêchent pas, mais vont les limiter en les rendant beaucoup plus difficiles à exploiter.

OpenWall. Il s'agit d'un patch développé par Solar Designer (entre autres créateur de popA3d, un serveur pop3 sécurisé) et prévu à l'origine pour les kernels 2.2.x (http://www.openwall.com/linux). Il a pour but de rendre les exploits de type stack overflow inopérants en rendant la stack non exécutable. Cela signifie que les shellcodes loadés en mémoire ne pourront plus être exécutés par le processeur si ils sont mis dans la pile. Malheureusement, il sera toujours possible d'exploiter ces failles en mettant le shellcode en-dehors de la pile, ou bien en utilisant la technique appelée "return into libc". L'option gcc trampoline va nous permettre de supporter les fonctionnalités de trampoline du compilateur.

Notez que sur les versions récentes de grsecurity, l'utilisation du patch openwall n'est plus proposée. En effet, c'est PaX qui se charger de la protection contre les buffers overflows (voir page suivante). Cependant, il faut savoir que plusieurs autres options de protection implémentées par grsecurity sont toujours tirées directement du travail de Solar Designer.

PaX. C'est un acronyme pour Page eXec (http://pageexec.virtualave.net). Il s'agit d'un autre patch de protection de la mémoire, beaucoup plus complet que le précédent. Attention, dans le cas où vous sélectionnez cette option, votre serveur X ne fonctionnera plus, ainsi que java et openoffice. L'utilisation d'un outil appelé chpax, en téléchargement sur http://pageexec.virtualave.net, sera nécessaire pour restaurer ces fonctionalités en désactivant la protection de PaX pour ces programmes.Dans un premier temps, PaX va rendre non exécutables les deux zones les plus critiques de la mémoire utilisateur que sont la stack (pile) et le heap (tas). Pour rappel, la stack est l'endroit où vont être stockées les variables initialisées ou non, tandis que le heap est l'endroit où va être stockée la mémoire allouée dynamiquement via les fonction de la famille de malloc(). PaX empêchant désormais l'exécution de code dans la stack et le heap, il propose en échange une émulation des gcc trampolines, mais l'auteur nous prévient que mettre cette option risquerait de provoquer un trou de sécurité dans le kernel. La libc standard propose une fonction nommée mprotect, qui a pour but de protéger une zone mémoire contre les accès, que ce soient en lecture, écriture ou encore exécution. PaX fournit une protection supplémentaire qui va permettre d'empêcher certains flags d'être utilisés. Par exemple, il sera impossible de rendre une page mémoire exécutable si elle ne l'était pas avant, ou encore de rendre inscriptible une page mémoire en lecture seule.

Randomize mmap() base. Si elles semblent être très restrictives, les protections de PaX seraient toutefois contournables via les exploits de type return into libc. Voir à ce sujet l'article "Defeating PaX protections", phrack 58 (http://www.phrack.org). Maintenant, **PaX** permet donc de rendre aéatoire l'emplacement en mémoire des libraries dynamiques, en modifiant l'appel système mmap(). Ainsi l'exploitation des buffer overflows deviendra quasiment impossible.

Read only kernel memory. Les travaux de Silvio Cesare (http://bignet.au/~silvio), puis, plus tard, de Sauron ou encore de Sd ("Kernel patching without lkms"du phrack issue 58), ont montré qu'il était possible de lire et écrire des données dans la mémoire du noyau, accessible à l'utilisateur par le device /dev/kmem. Les outils de type /dev/kmem sont beaucoup plus puissants que les simples lkm, puisqu'ils tapent directement dans la mémoire kernel. Ils sont aussi beaucoup plus discrets. GRsecurity va permettre de rendre cette mémoire read-only, au moins du point de vue userland. Cette option est parfaite pour un serveur si on pense à désactiver le support des modules dans le kernel: il devient ainsi quasiment impossible de cacher une backdoor ou autres dans le kernel.

Access Control List. Les ACL sont une série d'outils permettant de contrôler les accès au niveau utilisateurs à la fois sur les processus et sur les fichiers présents sur le système. Pour pouvoir utiliser les ACL, gradm, un programme disponible sur http://www.gresecurity.net est nécessaire. Il s'agit d'un programme fonctionnant en espace utilisateur et définissant les droits des différents utilisateurs pour la partie noyau des ACL GRsecurity. Les ACL sont un outil extrêmement puissant mais qui sont à manipuler avec une grande précaution si on en veut pas voir le système devenir parfaitement inutilisable. Il est par ailleurs fortement recommandé de mettre en place l'option denied capability logging, de manière à voir ce qui va et ne va pas dans la liste des ACL.

Filesystem protection

Cette section a pour but de protéger l'environnement de la machine contre des accès non autorisés.

Proc restriction. /proc est un répertoire dans lequel on trouve les informations en temps réel sur l'état du système, comme les processus en cours, les devices présents, etc. Proc restriction va limiter les accès à /proc notamment dans le listing des processus. Ainsi, ps n'affichera plus que les processus appartenant à l'utilisateur qui l'invoque, ou tous ceux n'appartenant pas au root. On pourra aussi choisir de restreindre l'accès à /proc pour un groupe précis. Pour mémoire, /proc est un répertoire vide tant que personne n'y entre, et dont les fichiers font 0 octets et sont vides tant qu'on ne cherche pas à y accéder. Cela permet donc d'avoir les informations sur le système en temps réel.

Linking restriction. Il s'agit là, d'une protection contre les exploits de type race condition. Un utilisateur ne pourra plus suivre un lien symbolique situe dans un répertoire ayant le stycky bit activé (+t) tel que /tmp, à moins qu'il ne soit le propriétaire du lien en question. Pareillement, un utilisateur ne pourra plus faire de hard links sur un fichier ne lui appartenant pas. Ceci va permettre de supprimer de nombreuses failles de sécurité locales.

FIFO restriction. Cette option va agir sur les fichiers de type FIFO (les "pipes") situés dans un répertoire ouvert en écriture à tout le monde (+t). Si cette option est activée, un utilisateur ne pourra pas écrire dans un fichier de type FIFO situé dans un tel répertoire, à moins d'être lui même propriétaire du répertoire.

Secure file descriptors. Cette option va permettre de sécuriser les descripteurs de fichiers correspondant à l'entrée standard, la sortie standard et la sortie d'erreur (0, 1 et 2). Si quelqu'un tente d'ouvrir un de ces descripteurs de fichiers, il sera immédiatement redirigé vers /dev/null. Cela permet ainsi d'éviter des failles comme l'exploit local des file descriptors sur les systèmes *BSD. Cette option n'est pas réellement utile et a été supprimée, car la bibliothèque de fonctions glibc intègre déjà cette protection.

Chroot jail restrictions. Pour mémoire, un chroot est une restriction de l'environnement à un répertoire donné. L'appel système chroot() est géré directement par le noyau. Restricted signals va empêcher un utilisateur chrooté d'envoyer des signaux en dehors de l'environnement restreint. Deny mounts va empêcher un utilisateur chrooté de monter, démonter ou remonter un système de fichiers, et ce même si le flag "user" est présent dans le fichier /etc/fstab. Deny double-chroots va empêcher un processeur de se chrooter dans un environnement chrooté. Il s'agit d'une méthode commune pour sortir d'un environnement chrooté. A ce sujet, voir les travaux de spacewalker sur le cassage des environnements chrootés sur les kernels 2.4.x. Enforce chdir(/) on all chroots va permettre de forcer le "/" sur le répertoire chrooté, ce qui permettra d'éviter certaines manœuvres de sorties de chroot. Deny (f)chmod +s: les programmes suid root peuvent toujours présenter un danger. Cette option va empêcher de donner le bit suid à des programmes qui ne l'ont pas encore au sein d'un environnement chrooté. Il s'agit d'une protection qui limitera les risques de bien des local root exploits. Deny mknod va empêcher la création d'inodes au sein d'un environnement chrooté (c'est à dire qu'il faudra lancer mknod lors de la préparation de l'environnement. Ceci fera d'ailleurs l'objet d'un article spécifique). Deny ptraces va empêcher l'utilisation de l'appel système ptrace() qui modifie le fonctionnement de certains programmes. Restrict priority changes va empêcher de changer la priorité d'un processus au sein de l'environnement chrooté. Cela va permettre de limiter l'utilisation des exploits sur des race conditions, qui ont parfois besoin d'augmenter leur priorité.

Kernel auditing. Cette section va permettre d'activer le logging sur un certain nombre d'événements à risques, voir critiques. Les options sont relativement explicites et nous ne les détaillerons pas ici. Il est simplement très recommandé d'activer Single group for auditing avant tout, de manière à centraliser les logs à un seul groupe. Il est aussi très recommandé, avant d'activer une option de log, de vérifier la place disponible sur le disque dur.

Executable protections

Exec process limiting. Le nombre de fork() autorisé par utilisateur est limité par le système. Mais pas le nombre de processus lancés à l'aide des appels systèmes de type exec* (execve(), evecle()...). Cette fonctionnalité permet cette limitation aussi. Les différentes limites sont définies dans /usr/include/limits.h.

Randomized PIDs. Lorsqu'une nouvelle tâche est lancée, un numéro de processus lui est attribué. L'activation de cette option va permettre de rendre aléatoires les numéros de pids, et va faire ce qu'il faut pour que les pids ne soient pas réutilisés trop vite. Cette option est en fait à cocher avec le proc restriction, pour empêcher à des utilisateurs de s'attaquer à des pids en local.

Altered default IPC permissions. Cette option va permettre de changer les permissions sur les IPC lancées par un utilisateur en prenant comme référence son umask, plutôt que d'utiliser les permissions de base de Linux qui sont ugo+rwx. Il faut noter que certains programmes ne fonctionneront plus si cette option est activée. C'est notamment le cas du serveur apache.

Limit uid/gid changes to root. Cette option va interdire à des utilisateurs normaux l'utilisation des appels system setuid(), setgid(), setduid() et setegid().

Fork-bomb protection. Le fork() bomb est une technique qui permet de faire un DoS en lançant un très grand nombre de forks sur un processus donné, de manière à bloquer toutes les ressources de la machine, et, finalement, la faire planter. La protection contre les fork bomb va limiter le nombre d'appels à fork() par seconde, pour un GID donné.

Trusted path execution. Cette option permet de restreindre l'exécution de programmes pour un groupe donné aux exécutables situés dans un répertoire appartenant au root, et avec les droits en écriture pour le root seulement (adieu les exploits locaux).

Network Protections

Randomized IP IDs. Cette option va remplir le champ ID des paquets IP de manière complètement aléatoire, de telle sorte que la machine ne soit pas utilisée comme bouncer pour certains types de portscanning, ou encore pour déjouer les outils d'OS fingerprinting comme queso (http://www.insecure.org/nmap). Pareillement, l'option altered ping IDs va changer l'ID des paquets icmp en y mettant l'ID de la machine émettrice du paquet pour déjouer les tentatives d'OS fingerprinting. Randomized TTL va changer le ttl des paquets de manière complètement aléatoire, là aussi pour déjouer les tentatives de détection d'OS distante.

Randomized TCP source ports. Lors d'un appel à la fonction connect(), le numéro du port utilisé sera choisi de manière totalement aléatoire (dans la liste des ports non privilégiés), au lieu d'utiliser l'algorithme d'incrémentation de la stack IP de GNU/Linux.

Randomized RPC XIDs. Ici, les XIDs des fonctions RPC (c'est-à-dire les programmes gérant NFS, ou encore certains programmes comme fam pour enlightenment 17) seront générés de manière totalement aléatoire.

Socket Restriction. Cette dernière option va permettre d'interdire l'utilisation des sockets à des groupes d'utilisateurs, en choisissant soit les sockets client, soit les sockets serveur, soit tous les types de sockets. Cette option est à manipuler avec précaution.

Sysctl support. Si cette option est activée, les options de GRsecurity vont pouvoir être redéfinies au boot en passant des paramètres au noyau avant de le lancer. Personnellement, je déconseille cette option si des utilisateurs peuvent avoir un accès physique à la machine (et donc rebooter). Même si le mot de passe bios au démarrage est activé.

Et pour finir...

Voilà pour la description des possibilités de GRsecurity. Il ne me reste plus qu'à vous expliquer comment l'appliquer sur les sources de votre kernel avant de lancer la configuration.

```
[root@toto:/usr/src] ln -s linux linux-2.4.18
[root@toto:/usr/src] patch -P0 < grsecurity-1.9.6-2.4.18.patch.gz
[root@toto:/usr/src] cd linux
[root@toto:/usr/src] make menuconfig
```

Comme nous l'avons vu, bien plus qu'un simple patch, GRsecurity est plutôt un amalgame de patches de sécurité permettant de contrôler un très grand nombre d'options à très bas niveau. Le nombre d'options est parfois un peu déroutant et on aimerait avoir une documentation un peu plus importante sur les diverses fonctionnalités. De très nombreuses features sont parfaitement inutiles dans le cadre d'une workstation, mais deviennent vite indispensable sur un serveur relié au net ou sur une machine de firewall.

En vente chez votre marchand de journaux