Protection des données stockées CLI Linux

J'ai déjà parlé d'outils qui permette de protéger vos données ici, Je voudrai faire un petit complément d'information.

Pourquoi, protéger vos données me rediriez-vous?
On peut mettre des droits particulier, sur nos fichiers et répertoires.
Les permissions standard sur le système de fichiers permettent de se protéger des accès inopportuns.
Un problème demeure toutefois, quiconque a un accès direct au disque dur depuis un système d'exploitation tiers, en remontant le disque dans une nouvelle machine par exemple ou plus simplement en démarrant sur un live-cd, peut accéder aux données en court-circuitant toutes les permissions en place sur les fichiers.

je vais cette fois vous proposez de le faire à différent niveaux fichiers, disque ou partition et filesystem.

1. Protection au niveau fichier

La solution la plus simple et la plus souple pour protéger un fichier sensible est de le chiffrer directement : un nouveau fichier chiffré est donc généré, à la place ou en plus du fichier d'origine.
Bon, j'avoue que cette parti je l'ai déjà abordé, je vais juste ajouté un outils en plus `crypt` (certes archaïque).
Penser a installer `dmcrypt` :
Alors, pour commencer il nous faut un fichier non chiffré.

```
~# echo ksh veux dire korn shell > public
```

On chiffre le contenu du fichier public et on envoie le résultat vers un fichier secret.
Notez les redirections d'entrées et de sorties standard.

```
~# crypt < public > secret
Unix crypt(1) emulation program using mcrypt(1).

Use crypt -h for more help.
Enter the passphrase (maximum of 512 characters)
Please use a combination of upper and lower case letters and numbers.
Enter passphrase:
Enter passphrase:

Stdin was encrypted.
```

On supprime le fichier non chiffré et on vérifie le caractère impénétrable du fichier protégé.

```
~# rm public
```

```
~# file secret
secret: data
~# cat secret
▒▒▒▒?]▒.▒)▒▒19▒▒▒▒[▒L▒▒
```

On déchiffre le contenu du fichier `secret` pour retrouvé le contenu en claire:

```
~# crypt < secret
Unix crypt(1) emulation program using mcrypt(1).

Use crypt -h for more help.
Enter the passphrase (maximum of 512 characters)
Please use a combination of upper and lower case letters and numbers.
Enter passphrase:
Enter passphrase:

ksh veux dire korn shell
Stdin was encrypted.
```

l’inconvénient de cette méthode est que l'utilisateur est obliger de déchiffrer manuellement le fichier à chaque fois qu'il veut consulter ou modifier les données. Sans oublier de le chiffrer à nouveau ensuite.

2. Protection au niveau disque ou partition

une autres solution est de gérer le chiffrage directement au niveau des blocs du disque.
Les accès au disque sont chiffrés au plus près du matérielle, le filesystem ordinaire utilisé provoque l'écriture des blocs de données qui sont alors chiffrés par le mécanisme de protection du disque.
L'accès est transparent pour l'utilisateur, il enregistre un fichier sur son disque protégé et le système se charge de chiffrer les blocs au moment de leur écriture et de les déchiffrer à la volée lors de leur lecture. On chiffre la totalité d'une ou de plusieurs partitions physiques ou logiques, mais en laissant toujours non chiffrée la partition hébergeant le noyau, Si vous voulez amorcer votre système.

Je vais utilisé l'outil `cryptsetup`, j'ai préparé la partition `sdb1` qu'on ne peut pas lire normalement.

2.1. Préparation de la partition ''sdb1'' :

```
~# cryptsetup luksFormat -c aes-xts-plain64 -s 512 -h sha512  /dev/sdb1
```

On invoque cette commande pour formater la partition au type LUKS (initialiser la partition LUKS et définir la clé initiale).

Le conteneur chiffré de manière standard va stocker la clé de chiffrement maître qui servira à ouvrir votre volume chiffré.
Il est possible d'ajouter jusqu'à 8 clefs supplémentaires dans des “slots”, qui déverrouillent l'accès à

la clef maître.
Vous pouvez ainsi avec cette méthode avoir plusieurs utilisateurs qui vont chacun ouvrir le conteneur chiffré avec leur clef et il vous sera possible de révoquer les clés éventuellement compromises.

```
~# mount /dev/sdb1 /mnt
mount : unknown filesystem type 'crypto_LUKS'
```

2.2. Montage manuel et formatage

On crée un fichier spécial de type bloc à partir de la partition chiffrée.
Il est nécessaire pour cela de renseigner la clé de chiffrement.
Ouverture et formatage en ext4 de la partition chiffrée.
L'appellation du volume est ici `perso`

```
cryptsetup luksOpen /dev/sdb1 perso
Entrez la phrase de passe pour /dev/sdb1 :
```

Formatage de la partition en `ext4` :

```
mkfs.ext4 /dev/mapper/perso
```

Puis montage de la partition :

```
mount -t ext4 /dev/mapper/perso /mnt/
```

Le filesystem monté de type ordinaire, ne permet pas de savoir qu'on est sur un support physique chiffré.

Démontage et fermeture du volume chiffré :

```
umount /mnt
cryptsetup luksClose perso
```

3. Protection au niveau filesystem

Le meilleur compromis entre souplesse et efficacité sera apporté par le chiffrement au niveau du filesystem.
On pourra limiter les données chiffrées à celles contenu dans un filesystem monté, ce qui, du point de vue de l'utilisateur, reviendra à disposer d'un répertoire chiffré.
La plupart des distributions Linux courantes proposent aujourd'hui de chiffrer le répertoire utilisateur `home`. La simplicité de mise en œuvre et d'exploitation des solutions de chiffrement au niveau filesystem justifie également leur popularité grandissante. Nous aborderons cette technologie par la solution la plus répandue `ecryptfs`.

Le principe est d'exploiter un répertoire généralement caché, généralement, pour y stocker les données chiffrés.
Ce répertoire sera monté comme un filesystem `ecryptfs`, le répertoire chiffré reste consultable, mais évidement sans intérêt sous sa forme direct.

Toute opération de montage ecryptfs doit être validé par un mot de passe.

Il faut installer le package `ecryptfs-utils`

Dans un premiers temps nous allons créer un répertoire de données :

```
~# mkdir .perso perso
```

Puis montage ecryptfs, vous devrez répondre à quelques question :

```
~# mount -t ecryptfs .perso /mnt
Select key type to use for newly created files:
  1) tspi
  2) passphrase
Selection: 2
Passphrase:
Select cipher:
  1) aes: blocksize = 16; min keysize = 16; max keysize = 32
  2) blowfish: blocksize = 8; min keysize = 16; max keysize = 56
  3) des3_ede: blocksize = 8; min keysize = 24; max keysize = 24
  4) twofish: blocksize = 16; min keysize = 16; max keysize = 32
  5) cast6: blocksize = 16; min keysize = 16; max keysize = 32
  6) cast5: blocksize = 8; min keysize = 5; max keysize = 16
Selection [aes]: 2
Select key bytes:
  1) 16
  2) 32
  3) 56
Selection [16]: 3
Enable plaintext passthrough (y/n) [n]:
Enable filename encryption (y/n) [n]:
Attempting to mount with the following options:
  ecryptfs_unlink_sigs
  ecryptfs_key_bytes=56
  ecryptfs_cipher=blowfish
  ecryptfs_sig=53013ee4f02a1993
WARNING: Based on the contents of [/root/.ecryptfs/sig-cache.txt],
it looks like you have never mounted with this key
before. This could mean that you have typed your
passphrase wrong.

Would you like to proceed with the mount (yes/no)? : yes
Would you like to append sig [53013ee4f02a1993] to
[/home/test/.ecryptfs/sig-cache.txt]
in order to avoid this warning in the future (yes/no)? : no
Not adding sig to user sig cache file; continuing with mount.
Mounted eCryptfs
```

Exploitation du filesystem :

```
~# cd /mnt
~# echo test > test
~# cat test
 test
```

Démontage et essai de visualisation :

```
~# umount /mnt
~# cd .perso
~# ls -l
total 12
-rw-r--r-- 1 test test 12288 nov.   7 22:34 test
~#  file test
test: data
~# cat test
x▒?▒!8
  ⊦@@▒
        =▒▒    r▒▒▒▒▒f▒▒3▒\▒/▒^▒▒▒▒▒0Pr!▒Z▒6^▒{M/▒CE▒d9▒▒▒▒/5e}▒▒%▒▒p3▒▒▒?▒▒▒▒
r▒▒▒□=▒▒▒▒▒U▒^N[▒▒▒▒▒▒▒▒j▒▒▒>▒K<D/▒/▒h▒▒▒▒6▒T▒M▒▒▒▒U▒
▒#▒arVP▒A▒=▒▒▒▒▒+▒▒▒h(,▒▒▒P▒▒▒I▒=▒▒▒▒▒▒|{▒8⊦▒▒▒▒H▒▒▒▒□6▒/&▒,
                                                        ▒▒▒n▒▒▒3
▒.▒┼▒P▒#▒▒▒▒▒▒▒@▒▒▒6▒▒▒▒>zS8p▒I3▒l▒D▒▒▒V(;▒▒▒▒>▒>▒MlI▒Sm$@H▒ÆXu▒^▒▒▒r,&▒▒▒p
鄙▒▒4▒4i▒▒▒y▒}▒V5S龕cX▒▒▒G▒▒▒d▒8[yu}
C+5▒%[▒5▒▒▒NU▒tRrϱ#▒▒;▒bo]J▒4▒$▒S▒▒▒▒'▒7▒j▒▒▒yK▒▒▒▒▒▒▒▒<▒▒▒▒G\▒k▒r
```

source : ubuntu-fr.org