Bisoños Usuarios de GNU/Linux de Mallorca y Alrededores | Bergantells Usuaris de GNU/Linux de Mallorca i Afegitons

Cifrando un sistema de ficheros (23229 lectures)

Per **Carles Pina i Estany**, ***cpina*** (http://pinux.info)

Creado el 30/01/2004 01:05 modificado el 30/01/2004 01:05

En Linux tenemos soporte para muchos sistemas de ficheros, y también para sistemas de ficheros tipo "loopback" (es como un sistema de ficheros dentro de un fichero).

También, y sobretodo a partir de la versión 2.6 del Kernel que ya viene incluido de serie, tenemos muy buen soporte para sistemas de criptografía.

En este "breve" veremos como podemos unir las dos partes: criptografía y sistemas de ficheros. Y todo sin tener que reparticionar. Veremos como podemos trabajar de forma normal y transparente en un fichero que estará cifrado.

Versió en català disponible aquí[1] (Catux)

Introducción

Con los sistemas de ficheros cifrados podremos montar un sistema de ficheros que se necesite contraseña para acceder en él. Sin la contraseña, nadie podrá ver su contenido (ni root ni nadie) ya que la información estará cifrada.

Aunque tenemos todo el sistema de permisos inherente de Linux (y cualquier Unix), eso puede ser útil por si nos roban físicamente el disco duro. O si somos algo descuidados y se puede arrancar mediante CD-ROM, pasarle parámetros a nuestro Kernel, etc.

Aquí explicaremos como usar los sistemas de ficheros cifrados en un Kernel 2.6.1 (aunque en cualquier 2.6.x debería ser muy parecido, hasta con los 2.4.x y con los parches de http://www.kerneli.org[2].

Compilación del Kernel

Supongamos que nos hemos bajado un Kernel 2.6.x y lo sabemos compilar de forma correcta.

Deberemos ir a la configuración del Kernel en el apartado `Cryptographic options` y activar los algoritmos de cifrado que deseemos. Útiles para hacer algunas pruebas podrian ser `NULL, MD5, DES and Triple DES, Blowfish` aunque podemos poner más. Los seleccionaremos, al menos de momento, todos como módulos.

También tenemos que ir en `Device Drivers`, después el menú `Block devices` y marcar la opción `Loopback device support` con la opción, como módulo de `Cryptoloop Support`.

Y herramientas de usuario

Necesitamos soporte para criptografía en al menos el paquete `mount` (lo estaremos usando con el losetup). El paquete de Debian no tiene el último parche para criptografía. Podemos descargarnos el paquete desde aquí[3]. En Gentoo parece que al hacer el emerge ya pone el parche para soporte de criptografía.

Como nota curiosa, con el paquete mount que viene en Debian Sid, no hay problema para cifrar con el algoritmo `blowfish` pero en cambio con otros algoritmos no podremos hacerlo (nos dará un error)

Montando un fichero... sin cifrar

Lo que haremos ahora es crear un fichero y montarlo a través del dispositivo de loopback como si fuera una partición. En el otro punto haremos lo mismo pero cifrando el fichero.

Los pasos para montar el fichero pasando por el dispositivo de loopback y **sin cifrar** serian:

`dd if=/dev/zero of=fichero.ext2 bs=1M count=100`
De esa forma hacemos un fichero "en blanco" de 100 MB.

`losetup /dev/loop0 fichero.ext2`
Ahora hemos asociado el `/dev/loop0` al `fichero.ext2`. Lo podemos comprobar si ejecutamos:

```
pinux:/mnt/dades# losetup /dev/loop0
/dev/loop0: [0308]:3 (fitxer.cripto)
```

Ahora podemos pensar que tenemos en `/dev/loop0` una "nueva partición". Por tanto podemos hacer: `mkfs -t ext2 /dev/loop0`
Y ya habremos formateado la "nueva partición".

Después, como siempre en las particiones podemos hacer: `mount -t ext2 /dev/loop0 prueba/`
Para comprobar que ha ido bien, ejecutamos `mount` y veremos una línea como:

```
/dev/loop0 on /mnt/dades/prueba type ext2 (rw)
```

Y podemos trabajar en él de forma normal. Cuando terminemos: umount /dev/loop0 losetup -d /dev/loop0 Es importante recordar el `losetup` final porqué sinó queda "medio desmontado".

Montando un fichero... cifrado

Antes de todo, cargaremos el módulo cryptoloop (`modprobe cryptoloop`). También cargaremos el sistema de cifrado que deseemos, por ejemplo `blowfish`, `des`, `twofish`, etc.

Para crear el fichero, esa vez es mejor hacerlo así:
`dd if=/dev/urandom of=fichero bs=1M count=10`
(de esa forma no será todo "nulos", sinó que contendrá datos aleatorios desde el principio)

Seguidamente haremos:
`losetup -e blowfish /dev/loop0 fichero.ext2`
Donde nos pedirá la contraseña para el sistema de ficheros.

Después lo podemos formatear como antes y montar de forma normal:

```
mkfs -t ext2 /dev/loop0
mount -t ext2 /dev/loop0 prueba/
```

Y a partir de aquí tenemos de forma fácil un sistema de ficheros, contenido en un fichero y cifrado. Podemos trabajar en él de forma totalmente normal.

Cuando terminemos:

```
umount /dev/loop0
losetup -d /dev/loop0
```

Como última nota, cuando lo volvamos a montar:

```
losetup -e blowfish /dev/loop0 fichero.ext2
mount -t ext2 /dev/loop0 prueba/
```

Si en `losetup` nos equivocamos de contraseña no se "quejará", pero el mount fallará (ya que al no poder descifrar el sistema de ficheros no puede acceder al superblock y otra información necesaria para montarla).

Al ejecutable losetup le podemos pasar el **algoritmo de cifrado** y también **el número de bits** que queremos que se usen para el cifrado. Por ejemplo, valores válidos pueden ser `des-64 3des_ede-192 blowfish-32 blowfish-256 twofish-256`, etc. Cuantos más bits de cifrado, el algoritmo es más seguro pero también más lento.

Lo hemos hecho con ext2 pero lo podemos hacer con cualquier otro sistema de ficheros. Pero si lo hicieramos con ReiserFS recordar que él ya usa 30MB para hacer el Journaling, que en una partición normalmente no tiene mayor importancia pero si estamos haciendo un fichero de 50MB o 100MB puede ser demasiado.

Hay información de los sistemas de cifrado (a medida que vamos cargando más módulos) en `/proc/crypto`

¿Y la velocidad?

Evidentemente, el hecho de estar cifrando y descifrando tiene su penalización de rendimiento. He hecho un pequeño "benchmark" (si se puede llamar así). El ordenador es un Pentium 3 a 1000 Mhz portátil (Compaq Presario 1700) y el disco duro responde de esa forma:

```
pinux:/mnt/dades# hdparm -t /dev/hda

/dev/hda:
 Timing buffered disk reads:   50 MB in  3.08 seconds =  16.22 MB/sec
 pinux:/mnt/dades# hdparm -T /dev/hda

/dev/hda:
 Timing buffer-cache reads:   756 MB in  2.00 seconds = 377.11 MB/sec
 pinux:/mnt/dades#
```

Copiar un fichero de 67 MB sin usar el `losetup` tarda unos 19 segundos (origen y destino en el mismo disco duro, diferentes particiones).

Copiarlo dentro de un fichero con losetup, tarda más o menos lo mismo (unos 19-20 segundos).

Ya por último, en des-64, blowfish-32, blowfish-256, twofish-256, etc. tarda aproximadamente lo mismo (20-21 segundos; puede llegar a 22-23 segundos con twofish).

Y con 3des_ede-192 tarda unos 31 segundos aproximadamente (un 50% más que los otros).

La prueba es tan simple que no se pueden sacar conclusiones más allá que para un uso normal de los datos no penaliza mucho. Por ejemplo, se podria tener sin mucho problema el ~/mail y algun otro directorio montados de esa forma y trabajar de forma habitual.

Lista de enlaces de este artículo:

1. http://catux.org/index.php?contingut=articles&menu=6&num=36
2. http://www.kerneli.org
3. http://www.inittab.de/debian/util-linux/crypto-loop/

E-mail del autor: carles _ARROBA_ pinux.info
Podrás encontrar este artículo e información adicional en: http://bulma.net/body.phtml?nIdNoticia=1970