
# ISEC Lab #1

# Particiones de Disco Cifradas

Vicente Aguilera Díaz
vaguilera<arroba>isecauditors.com

# 1 Introducción

En este artículo presento una de las alternativas que podemos utilizar para la protección, mediante cifrado, de los datos que almacenamos en nuestras particiones de disco. La solución que presentaré se basa en el paquete loop-AES (loop-AES.sourceforge.net) aplicado sobre un sistema Linux RedHat9.

loop-AES trabaja utilizando dispositivos loop (loop devices). Un dispositivo loop es un dispositivo orientado a bloque que no almacena en sí ningún dato sino que redirige todas las operaciones de lectura y escritura al dispositivo o fichero que reside por debajo de él. Lo podemos ver como una capa intermedia entre los programas que realizan operaciones de E/S y los propios dispositivos o ficheros que almacenan los datos. Entonces, nos preguntaremos, ¿qué nos aporta un dispositivo loop?, pues bien, el loop device que utilizaremos en este caso y ya veremos porque digo en este caso, se encargará de cifrar y descifrar los datos que se almacenan en los dispositivos o ficheros de almacenamiento. Por defecto Linux proporciona un conjunto de dispositivos loop que podemos ver en el directorio /dev. Cada uno de estos dispositivos loop puede ser vinculado a un dispositivo de bloque o fichero real. Es decisión nuestra determinar que dispositivo loop vamos a utilizar. Los pasos que mostraremos utilizarán el dispositivo /dev/loop0.

Otro aspecto que aparecerá al configurar la partición cifrada es el tema de los ficheros de clave en modo single-key y en modo multi-key. El algoritmo AES soporta dos modos de clave: single-key y multi-key. El primero como su nombre indica utiliza un vector IV (Init Vector) simple y una sola clave de cifrado/descifrado para todos los sectores del dispositivo loop. En cambio el modo multi-key utiliza un IV más seguro utilizando MD5 y 64 claves AES diferentes para cifrar/descifrar los sectores en el dispositivo loop. La primera clave se utiliza para el primer sector, la segunda clave para el segundo sector, etc..

La primera utilidad que vamos a ver es **losetup**. Este comando permite vincular un dispositivo loop con un dispositivo de bloque o fichero. No necesitamos especificar ningún sistema de ficheros, esto se determina automáticamente o se especifica si lo deseamos a la hora de montar el sistema utilizando el comando **mount**. Un ejemplo sería el montaje de una imagen ISO de un CD-ROM que nos acabamos de bajar de Internet y queremos ver su contenido. Ejecutaríamos los siguientes comandos:

```
losetup /dev/loop0 freebsd-cd1.iso
mount -t iso9660 /dev/loop0 /mnt/cdrom
```

Ahora que ya conocemos el mecanismo que utiliza loop-AES para trabajar, veamos que necesitamos instalar en nuestro sistema Linux. Loop-AES proporciona un módulo para el kernel de Linux que es una reimplementación del módulo loop.o incorporando el algoritmo de cifrado AES. Este algoritmo de cifrado simétrico se puede utilizar para cifrar/descifrar sistemas de ficheros locales o particiones de disco.

# 2 Compilación de loop.o

El paquete loop-AES en sí no modifica nuestro kernel ya que trabaja con un módulo loop propio. Por tanto, vamos a explicar como debemos compilar el módulo loop.o.

Antes de compilar loop.o debemos asegurarnos de que nuestro kernel tiene las siguientes opciones:

```
CONFIG_MODULES=y
CONFIG_BLK_DEV_LOOP=n
CONFIG_KMOD=y
```

Si nuestro kernel tiene la opción CONFIG_BLK_DEV_LOOP=y o CONFIG_BLK_DEV_LOOP=m, no funcionará la instalación de loop-AES.

Si algunas de estas opciones no coincide, implicará que debemos recompilar nuestro kernel (siempre es buen momento para actualizar a la última versión).

Una vez tengamos el kernel compilado, no borraremos o moveremos los fuentes de sitio, ya que los necesitaremos al compilar el módulo loop.

Una vez tenemos el kernel con las opciones necesarias, podemos ejecutar la compilación:

```
make clean
make
```

La compilación e instalación de loop.o implica que se copie loop.o en el directorio /lib/modules/`uname -r `/block.

No es necesario añadir nada al fichero /etc/modules.conf.

El fichero Makefile intenta encontrar el directorio con el código fuente del kernel que se está ejecutando (si hemos cambiado la versión del kernel al recompilar será conveniente reiniciar el sistema con el nuevo kernel). Los directorios donde busca son:

/lib/modules/`uname -r`/build
/usr/src/linux
/usr/src/linux-`uname -r`
/usr/src/kernel-source-`uname -r`

Si queremos especificar el directorio de forma manual lo podemos hacer con el parámetro LINUX_SOURCE=/usr/src/linux-2.4.22aa1

# 3 Parcheando mount, umount, losetup, swapon y swapoff

Para dar soporte al algoritmo AES y a otros algoritmos, debemos parchear los comandos mount, umount, losetup, swapon y swapoff y recompilarlos. Con el paquete loop-AES se incluye el parche necesario y el paquete con estos comandos se llama util-linux y lo podemos descargar de ftp://ftp.win.tue.nl/pub/linux-local/utils/util-linux/.

La versión que utilizaremos será **util-linux-2.12a**. Siempre es recomendable verificar el hash MD5 del paquete.

Los pasos necesarios para parchear e instalar sólo los comandos mount, umount, losetup, swapon, swapoff (el paquete util-linux incorpora otros comandos que no vamos a tocar) son:

```
zcat util-linux-2.12a.tar.gz | tar xvf -
cd util-linux-2.12a
patch -p1 <../util-linux-2.12a.diff
CFLAGS=-O2 ./configure
make SUBDIRS="lib mount"
cd mount
install -m 4755 -o root mount umount /bin
install -m 755 losetup swapon /sbin
rm -f /sbin/swapoff && ( cd /sbin && ln -s swapon swapoff )
rm -f  /usr/share/man/man8/{mount,umount,losetup,swapon,swapoff}.8.gz
install -m 644 mount.8 umount.8 losetup.8 /usr/share/man/man8
install -m 644 swapon.8 swapoff.8 /usr/share/man/man8
rm -f /usr/share/man/man5/fstab.5.gz
install -m 644 fstab.5 /usr/share/man/man5
mandb
cd ../..
```

Ahora ya tenemos el módulo loop.o instalado y los comandos mount, umoun, losetup, swapon y swapoff con soporte para utilizarlo. ¿Qué más nos falta?

Si queremos hacer un test del driver loop.o podemos hacerlo ejecutando el comando make test. En caso contrario o después de realizar el test ya sólo nos queda  crear una partición cifrada para nuestros datos más críticos.

# 4 Creación de una partición cifrada

Para crear nuestra partición cifrada partiremos de los siguientes supuestos:

- Tenemos una partición /dev/hda6 vacía y de 20Gbytes
- La partición se montará en el directorio /mnt/encrypted
- Tenemos instalado gpg (GnuPG)
- Tenemos una sesión abierta con el usuario root.

Para el cifrado/descifrado de los datos utilizaremos 64 claves aleatorias(multi-key) que estarán cifradas (protegidas) a su vez con gpg. Debemos remarcar que la seguridad del sistema que vamos a montar radica

en el fichero con las 64 claves y también en las claves gpg que utilizamos para proteger este fichero. No estaría mal almacenar este fichero en algún medio removible como un token USB, una smartcard o algo parecido. Y por supuesto no hablaremos de la gestión de backups.

Cuando necesitemos montar nuestra partición deberemos proporcionar la contraseña que protege el fichero de claves, de manera que se obtengan las claves para cifrar/descifrar los datos en la partición.

El primer paso será crear el fichero de claves utilizando gpg y leyendo los datos aleatorios a partir del dispositivo /dev/random, recordemos que estas claves son simétricas:

```
head -c 2880 /dev/random | uuencode -m - | head -n 65 | tail -n 64 | gpg --
symmetric -a >/root/keyfile.gpg
```

Del comando anterior, podemos ver como el comando tail ejecutado antes de gpg extrae las últimas 64 líneas de datos que suponen las 64 claves aleatorias que vamos a utilizar para cifrar y descifrar los datos en la partición.

La lectura de datos aleatorios puede hacerse indefinida si el pool de entropía que tiene el kernel se queda vacío. Es recomendable que realicemos otras tareas mientras se generan las claves (utilizar el teclado, ratón, discos, etc.).

El siguiente paso será llenar la partición con datos aleatorios. Esto evita que se pueda identificar la parte de disco que tiene datos de la que está vacía (si por ejemplo rellenásemos la partición con la salida del dispositivo /dev/zero, la partición quedaría llena de ceros y con un simple análisis de la partición podríamos identificar las partes de disco que contienen la información cifrada). Esta operación puede llevar tiempo en función del tamaño de la partición que estemos utilizando.

```
head -c 15 /dev/urandom | uuencode -m - | head -n 2 | tail -n 1 | losetup -p 0
-e AES128 /dev/loop0 /dev/hda6

dd if=/dev/zero of=/dev/loop0 bs=4k conv=notrunc 2>/dev/null

losetup -d /dev/loop0
```

El primer comando genera los datos aleatorios para crear un password que no necesitamos recordar o guardar ya que se utilizará sólo para llenar la partición con datos incoherentes. A continuación define un vínculo entre el dispositivo loop0 y la partición cifrada utilizando el algoritmo de cifrado AES128. A diferencia del dispositivo /dev/random, /dev/urandom devuelve tantos datos como son solicitados aunque el pool de entropía del kernel esté vacío. Desde el punto de vista criptográfico es más seguro utilizar /dev/random pero dentro del propósito de este comando nos es suficiente con utilizar /dev/urandom.

El segundo comando escribe la salida del dispositivo /dev/zero, pasándolo a través del dispositivo loop0 en la partición cifrada. Con este comando

conseguimos que todos los valores cero que salen de /dev/zero sean cifrados (convertidos en datos incoherentes) y escritos en /dev/hda6.

Una vez hemos acabado el tercer comando elimina el vínculo entre /dev/loop0 y /dev/hda6

El último paso que necesitaremos realizar será crear un sistema de ficheros en nuestra partición cifrada para comenzar a utilizarla:

```
losetup -e AES128 -K /root/keyfile.gpg /dev/loop0 /dev/hda6
mkfs -t ext3 /dev/loop0
losetup -d /dev/loop0
```

Ahora ya tenemos la partición creada y la podemos montar. La forma sencilla es mediante el comando mount:

```
/sbin/losetup -e AES128 -K /root/keyfile.gpg /dev/loop0 /dev/hda6
mount -t ext3 /dev/loop0 /mnt/encrypted
```

Esto lo podemos automatizar añadiendo al fichero /etc/fstab la entrada correspondiente:

```
/dev/hda6     /mnt/encrypted      ext3
defaults,loop=/dev/loop0,encryption=AES128,gpgkey=/root/keyfile.gpg 0 0
```

De esta forma podemos utilizar el comando:

```
mount /mnt/encrypted
```

mount se encarga de solicitarnos la contraseña para desbloquear el fichero de claves.

Si añadimos al fichero /etc/fstab una entrada para nuestra nueva partición cifrada y hacemos que esta partición sea montada cada vez que se inicia el sistema, durante el inicio de éste se solicitará la contraseña para desproteger el fichero de claves. Esto puede ser una desventaja por el hecho de que en primer lugar la partición no tiene porqué montarse a no ser que sea necesario utilizarla y también por el hecho de no poder reiniciar el sistema de forma desatendida.

Una solución alternativa a esto podría ser configurar el inicio de la partición como un servicio independiente. Un ejemplo de script init.d para crear un servicio independiente sería:

```
#!/bin/bash
#

start() {
        echo -n $"Mounting /mnt/encrypted: "
        /sbin/losetup -e AES128 -K /root/keyfile.gpg /dev/loop0 /dev/hda6
        mount -t ext3 /dev/loop0 /mnt/encrypted
        RETVAL=$?
        echo
        [ $RETVAL = 0 ] && echo "Encrypted Partition mounted OK"
        return $RETVAL
```

```
}
stop() {
        echo -n $"Stopping $prog: "
        umount /mnt/encrypted
        /sbin/losetup -d /dev/loop0
        RETVAL=$?
        echo
        [ $RETVAL = 0 ] && echo "Encrypted Partition umounted OK"
}

# See how we were called.
case "$1" in
  start)
        start
        ;;
  stop)
        stop
        ;;
  status)
        mount | grep loop0
        losetup -a
        RETVAL=$?
        ;;
  restart)
        stop
        start
        ;;
  *)
        echo $"Usage: $prog {start|stop|restart|status}"
        exit 1
esac

exit $RETVAL
```

# 5 Enlaces de Interés

loop-AES: http://loop-aes.sourceforge.net
util-linux: ftp://ftp.win.tue.nl/pub/linux-local/utils/util-linux/
GnuPG: www.gnupg.org