HIDE STUFF - ENCRYPTED DEVICES

How do you keep your secrets safe on linux? How do you make sure that what you want no-one else to see can really not be seen by anyone else? Well linux allows you to create loopback devices using dm-crypt (get any files here). These are files which can be mounted and used as if they were disks. Linux also allows you to add encryption to these files, as well as any normal disk partitions. Lets take a look..

What is needed
I ran these tests on a fairly recent kernel, if you can try to run this on kernel 2.6.16 or better as previous versions had problems which effected the security of the dm-crypt process. The following kernel options could well be set by default but just make sure..

```
        * Code maturity level options
             o Prompt for development and/or incomplete code/drivers
        * Device Drivers -> Multi-device support (RAID and LVM)
             o Device mapper support
             o Crypt target support
        * Cryptographic options
             o AES cipher algorithms
```

Creating the loopback device
The loopback device is just a file of any size you want, remember? So lets make a file..

```
[root@localhost sandbox]# dd if=/dev/zero of=crypt.loop bs=1024 count=51200
51200+0 records in
51200+0 records out
[root@localhost sandbox]# ls -l
total 54260
-rw-r--r--  1 root root 52428800 Jun 21 22:27 crypt.loop
[root@localhost sandbox]#
```

What this did was create a 50MB file called *crypt.loop*, this is what we will use for our examples. On an added note, if you wanted to be a bit more paranoid and had the time, you can replace */dev/zero* with */dev/urandom*. Now lets map our file to a device..

```
[root@localhost sandbox]# losetup /dev/loop0 ./crypt.loop
```

There we go, now our file is mapped to */dev/loop0*.

Encrypting the loopback device
Ok, we have our device but we want to encrypt it as securely as possible, so..

```
[root@localhost sandbox]# cryptsetup -c aes-cbc-essiv:sha256 -y -s 256 luksFormat /dev/loop0
```

Now going into the full differences between the various crypto options would take to long here, suffice to say we are using the nice new versions (*-c aes-cbc-essiv:sha256*) with a key length of 256 bits (*-s 256*), the *-y* switch tells the encryption process to ask us for the key twice. Once you push enter you should see something similar to..

```
WARNING!
========
This will overwrite data on /dev/loop0 irrevocably.

Are you sure? (Type uppercase yes): YES
Enter LUKS passphrase:
Verify passphrase:
```

When you enter the passphrase it does not echo to the screen, thats why you do not see anything, but please make sure that you use a decent length passphrase so that you do not add in a weak link into the securing process of the device.

Setup the loopback device for use
So far so good, now lets setup the device for actual usage on our system. The first step is one you will also need to

do to access the encrypted device..

```
[root@localhost sandbox]# cryptsetup luksOpen /dev/loop0 crypt
Enter LUKS passphrase:
key slot 0 unlocked.
```

Doing this "unlocks" the device so that we can access it and maps it to the stated point under the */dev/mapper* structure, in this case *crypt*, but before anything further lets create a filesystem as per normal..

```
[root@localhost sandbox]# mkfs -j /dev/mapper/crypt
mke2fs 1.37 (21-Mar-2005)
Filesystem label=
OS type: Linux
Block size=1024 (log=0)
Fragment size=1024 (log=0)
12600 inodes, 50192 blocks
2509 blocks (5.00%) reserved for the super user
First data block=1
Maximum filesystem blocks=51642368
7 block groups
8192 blocks per group, 8192 fragments per group
1800 inodes per group
Superblock backups stored on blocks:
        8193, 24577, 40961

Writing inode tables: done
Creating journal (4096 blocks): done
Writing superblocks and filesystem accounting information: done

This filesystem will be automatically checked every 33 mounts or
180 days, whichever comes first.  Use tune2fs -c or -i to override.
```

Now lets use the loopback
We have setup the crypto options and created the filesystem, we have already unlocked the crypto device, so lets mount it..

```
[root@localhost sandbox]# mount /dev/mapper/crypt /mnt/test
```

Simple. Exactly as you would expect, lets see how it looks..

```
[root@localhost sandbox]# ls -l /mnt/test
total 12
drwx------  2 root root 12288 Jun 21 22:33 lost+found
[root@localhost sandbox]# df -h
Filesystem            Size  Used Avail Use% Mounted on
/dev/hda1             106G   89G   13G  89% /
/dev/shm               62M     0   62M   0% /dev/shm
/dev/mapper/crypt      48M  4.8M   41M  11% /mnt/test
```

Again, exactly as you would expect. Once you have finished with it, you just need to tidy up..

```
[root@localhost sandbox]# umount /mnt/test
[root@localhost sandbox]# cryptsetup luksClose crypt
[root@localhost sandbox]# losetup -d /dev/loop0
```

Working with disks
Now this process can be used with any normal disk partition, take a look at this..

```
[root@localhost mnt]# dd if=/dev/zero of=/dev/sda1 bs=1M
[root@localhost mnt]# cryptsetup -c aes-cbc-essiv:sha256 -y -s 256 luksFormat /dev/sda1
[root@localhost mnt]# cryptsetup luksOpen /dev/sda1 sda1
Enter LUKS passphrase:
key slot 0 unlocked.
[root@localhost mnt]# mkfs -j -c /dev/mapper/sda1
[root@localhost mnt]# cryptsetup luksClose sda1
```

So now this has created our encrypted device, given it a passphrase and created the filesystem. Great, but how do

we use it? First off, remember that it will only mount if you enter the passphrase, so if you restart your machine you will need to enter it again. So for a startup script take a look at this one..

```
PARTS=sda1
CRYPT=/sbin/cryptsetup

for CHECK in $PARTS
 do

echo "DOING ENCRYPTED PARTITION $CHECK"
MNTED=`df | grep $CHECK 1> /dev/null ; echo $?`

if [ $MNTED == "0" ]
 then
  umount /dev/mapper/$CHECK
fi
if [ -b /dev/mapper/$CHECK ]
 then
  $CRYPT luksClose $CHECK
fi
if [ ! -b /mnt/$CHECK ]
 then
  mkdir -p /mnt/$CHECK
fi

read -s -t 60 -p "-What is passphrase for $CHECK:  " RESULT
echo $RESULT | $CRYPT luksOpen /dev/$CHECK $CHECK
mount /dev/mapper/$CHECK /mnt/$CHECK

done
```

This script will run through any encrypted partitions you have (I only have one) , it checks to see if it's already mounted or open. Then it prompts for the passphrase and will time out if nothing is entered for 60 seconds. If it gets the correct passphrase it will mount the partition. As always, my scripts work for me, please feel free to use it and modify as needed. Putting this script into your runlevel startups will cause it to run upon startup, but if you miss the startup sequence, you can manually run the script to mount your partitions as well - works for me.

Final Words

Remember that encryption can be a very effective layer in your defense in depth strategy, these techniques particularly protect against some physical attacks or any digital attacks which will require or cause a restart. This functions as a fail-closed option rather then a fail-open. There are a lot more options to these commands, you can specify different encryption mechanisms, you can specify multiple passphrases, and more. As always, have fun and learn.