How to full encrypt your linux system with lvm on luks - Linux.com

Egidio Docile ⋮ 6-7 minutes ⋮ 5/18/2015

Security and privacy are two very important subjects, and everyone of us, in a way or another, has sensitive data stored on his computer. While you can consider pretty safe your data on a home computer, on a laptop (or any portable device) the situation is a lot different. You carry your device with you and don't want to loose all your precious data in case it is stolen or lost. Here is when system encryption comes in handy.

In this tutorial i will show you how to full encrypt your system using two linux native tools: LVM (for partitioning) and LUKS (for the actual encryption).

Why LVM on LUKS?

Imagine you have your hard drive divided in at least two partitions: one for the root of your system and the other used as a swap partition. You could encrypt them separately but then 2 passwords will be asked during boot time, and this is really annoying.

You could decide to avoid the use of swap partition or to use a random generated key, but in both cases you would lost the ability to hibernate (actually to resume from hibernation).

The solution is to use LVM partitioning: we will encrypt the whole disk with LUKS, then we will use the disk as phisical volume and make it part of a volume group which will contain as much logical volumes as we need, each for every partitions we want. The only partition that must be unencrypted is the boot partition, so for the most secure setup, we will use an external device for it. Using the LVM partitioning we won't even need to create a partition table, we will use the raw disk instead.

Why do it from command line?

Most of the modern distributions installers offer the option to encrypt the disk grephically, so why do it from CLI? Well, the answer is in two words: more control. Most graphical installers offer no chance to fine tune the encryption options, and none of them (that i know) would encrypt the whole raw disk without creating a partition table. Sure this could have no importance for you, and in this case you can use the default (and usually good) options. Anyway, it's always nice to know how things work under the hood.

First things first: fill your disk with random data

Before anything else, we need to fill the disk with random data, so that the whole device content would appear the same and indistinguishable, and no patterns could be discovered on it (for example what zones of the disk are filled with data and what are empty). Filling a disk with random data can be very time consuming, especially on very large hard drives, but we can use a trick here: we will luks format the device first, and then fill it with zeros (zeros are much faster to generate then random). Because of encryption the data will be written on the disk as random, so we're actually using the luks device as a random data generator device. At that point only the luks header will remain as clear data at the beginning of the disk and we will override it with random data from /dev/urandom.

Here's the process in few steps:

1) Create luks partition

```
cryptsetup luksFormat --hash=sha512 --key-size=512 --cipher=aes-xts-plain64 --
verify-passphrase /dev/sda
```

Note that obviously you can use different settings for the luksFormat command; above it's what i usually use. After that you will be asked to enter a password for the encryption, it doesn't matter if it's not very secure now, because we will only use this device as random data generator.

2) Open the encrypted device: the command below opens the luks device and maps it as "sda_crypt"

```
cryptsetup luksOpen /dev/sda sda_crypt
```

3) Now we fill this device with 0s using dd and /dev/zero as source:

```
dd if=/dev/zero of=/dev/mapper/sda_crypt bs=1M
```

4) All the underlying disk appears now to be filled with random data, minus the luks header that we are about to override (you can take a look using "hexdump /dev/sda | less" command). Usually the header takes few Megabytes, but to avoid calculations and be rude we will cover the first 10 Megabytes of the disk. We will use dd with /dev/urandom as random data source this time:

```
# first destroy the mapping
cryptsetup luksClose sda_crypt

# override the header
dd if=/dev/urandom of=/dev/sda bs=512 count=20480
```

5) We have now the disk full of random data. Now for the serious stuff. Just repeat steps 1 and 2 but this time use a very secure passhrase, because it will be the key to unlock your disk

6) Now we will use the device as phisical volume…

```
pvcreate /dev/mapper/sda_crypt
```

7) Now create a volume group (i will name it "vg00") that will contain the phisical device /dev/mapper/sda_crypt

```
vgcreate vg00 /dev/mapper/sda_crypt
```

8) Create the logical volumes. I usually use 4: one for root, one for the swap partition, one for /home and the other for a data partition, but this is obviously up to you. The "+100%FREE" options on the last line modifies the command to use logical extents instead of size, and to use all of the free remaining ones for that logical volume.

```
lvcreate -n lv00_swap -L 4G vg00
lvcreate -n lv01_root -L 30G vg00
lvcreate -n lv02_home -L 10G vg00
lvcreate -n lv03_data -l +100%FREE vg00
```

7) Now create the boot partition on a separate device, ideally an usb stick, and install grub on the mbr of this device. With this setup we both will have no clear partitions on our encrypted disk, and no chance to boot the system without the external device, which adds an extra layer of security.

Please rembember that encryption protects your data only on a pre-boot situation when the machine is not on. After you boot and decrypt the disk you will have no added protection. All you have to do now is to install your system as always, and enjoy full disk encryption!