Search

Configuring LUKS: Linux Unified Key Setup

Learn how to encrypt Linux partitions with the Linux Unified Key Setup (LUKS).

Posted: October 8, 2019 | 7 min read | Valentin Bajrami (Accelerator, Alumni)

Image by Hebi B. from Pixabay

According to Wikipedia, the Linux Unified Key Setup (LUKS) is a disk encryption specification created by Clemens Fruhwirth in 2004 and was originally intended for Linux. LUKS uses device mapper crypt (`dm-crypt`) as a kernel module to handle encryption on the block device level.

More Linux resources

- Advanced Linux commands cheat sheet
- Download RHEL 9 at no charge through the Red Hat Developer program
- A guide to installing applications on Linux
- Linux system administration skills assessment

There are different front-end tools developed to encrypt Linux partitions, whether they're plain partitions or Logical Volumes (LVs). In this tutorial, we'll explore these tools and demonstrate how to configure disk encryption. I've created a 10GB disk (`/dev/vdb`) to use during this tutorial.

Installing the tools

Let's start by installing the appropriate tools for configuring encryption:

```
dnf install -y cryptsetup parted
```

The `cryptsetup` package provides the `cryptsetup` command, which we'll use to configure encryption, while the `parted` package provides the `parted` command for configuring the partition.

Creating the partition

Running the `lsblk` command shows your current setup:

```
[root@rhel8 ~]# lsblk
NAME          MAJ:MIN RM  SIZE RO TYPE MOUNTPOINT
sr0            11:0    1 1024M  0 rom
vda           252:0    0   30G  0 disk
├─vda1        252:1    0    1G  0 part /boot
└─vda2        252:2    0   29G  0 part
  ├─rhel-root 253:0    0 26.9G  0 lvm  /
  └─rhel-swap 253:1    0  2.1G  0 lvm  [SWAP]
vdb           252:16   0   10G  0 disk
```

We can encrypt a whole block device like `/dev/vdb`, but creating a partition offers more flexibility since we can add other partitions later on.

Now we run the following commands to create a partition to encrypt:

```
[root@rhel8 ~]# parted /dev/vdb mklabel msdos
Information: You may need to update /etc/fstab.

[root@rhel8 ~]# parted /dev/vdb -s "mkpart primary 2048s -1"
[root@rhel8 ~]# parted /dev/vdb align-check optimal 1
1 aligned
```

When running `lsblk` again, we see that the `dev/vdb1` partition was added:

```
[root@rhel8 ~]# lsblk
NAME          MAJ:MIN RM  SIZE RO TYPE MOUNTPOINT
sr0            11:0    1 1024M  0 rom
vda           252:0    0   30G  0 disk
├─vda1        252:1    0    1G  0 part /boot
└─vda2        252:2    0   29G  0 part
  ├─rhel-root 253:0    0 26.9G  0 lvm  /
  └─rhel-swap 253:1    0  2.1G  0 lvm  [SWAP]
vdb           252:16   0   10G  0 disk
└─vdb1        252:17   0   10G  0 part
```

Formatting the volume with LUKS

The following process encrypts `dev/vdb1`. In order to proceed, you need to enter `YES` in capitals and provide the password twice:

```
[root@rhel8 ~]# cryptsetup -y -v luksFormat /dev/vdb1

WARNING!
========
This will overwrite data on /dev/vdb1 irrevocably.

Are you sure? (Type uppercase yes): YES
Enter passphrase for /dev/vdb1:
Verify passphrase:
Key slot 0 created.
Command successful.
```

Then, we need a target to open the encrypted volume. I used `mybackup` as my target, but this target can be named anything:

```
[root@rhel8 ~]# cryptsetup -v luksOpen /dev/vdb1 mybackup
Enter passphrase for /dev/vdb1:
Key slot 0 unlocked.
Command successful.
```

Running `lsblk` once again, we see:

```
[root@rhel8 ~]# lsblk
NAME          MAJ:MIN RM   SIZE RO TYPE  MOUNTPOINT
sr0            11:0    1 1024M  0 rom
vda           252:0    0   30G  0 disk
├─vda1        252:1    0    1G  0 part  /boot
└─vda2        252:2    0   29G  0 part
  ├─rhel-root 253:0    0 26.9G  0 lvm   /
  └─rhel-swap 253:1    0  2.1G  0 lvm   [SWAP]
vdb           252:16   0   10G  0 disk
└─vdb1        252:17   0   10G  0 part

└─mybackup 253:2 0 10G 0 crypt
```

We can also see the `mybackup` encrypted volume's mapping:

```
[root@rhel8 ~]# ls -l /dev/mapper/mybackup
lrwxrwxrwx. 1 root root 7 Sep 16 16:10 /dev/mapper/mybackup → ../dm-2
```

Creating a filesystem

Since we now can access the encrypted volume, we need to format it before we can store data on it. You can choose between different filesystem types, like xfs (the default on Red Hat Enterprise Linux 8), ext3, ext4, etc. For the sake of simplicity, we'll use xfs as the filesystem type:

```
[root@rhel8 ~]# mkfs.xfs /dev/mapper/mybackup

meta-data=/dev/mapper/mybackup   isize=512    agcount=4, agsize=654720 blks
         =                       sectsz=512   attr=2, projid32bit=1
         =                       crc=1        finobt=1, sparse=1, rmapbt=0
         =                       reflink=1
data     =                       bsize=4096   blocks=2618880, imaxpct=25
         =                       sunit=0      swidth=0 blks
naming   =version 2              bsize=4096   ascii-ci=0, ftype=1
log      =internal log           bsize=4096   blocks=2560, version=2
         =                       sectsz=512   sunit=0 blks, lazy-count=1
realtime =none                   extsz=4096   blocks=0, rtextents=0
```

Creating the mount point and directory

To write data on the encrypted filesystem, we need to mount it first. I chose `/mnt/my_encrypted_backup` to be the mount point for my data:

```
[root@rhel8 ~]# mkdir -p /mnt/my_encrypted_backup
```

Then we run the `mount` command:

```
[root@rhel8 ~]# mount -v /dev/mapper/mybackup /mnt/my_encrypted_backup/

mount: /mnt/my_encrypted_backup does not contain SELinux labels.
       You just mounted an file system that supports labels which does not
       contain labels, onto an SELinux box. It is likely that confined
       applications will generate AVC messages and not be allowed access to
       this file system.  For more details see restorecon(8) and mount(8).
mount: /dev/mapper/mybackup mounted on /mnt/my_encrypted_backup.
```

Here we get a Security-Enhanced Linux (SELinux) warning. We need to relabel the mount point's SELinux security context:

```
[root@rhel8 ~]# restorecon -vvRF /mnt/my_encrypted_backup/

Relabeled /mnt/my_encrypted_backup from system_u:object_r:unlabeled_t:s0 to
system_u:object_r:mnt_t:s0
```

Running the `mount` command once again shows that the warning is gone:

```
[root@rhel8 ~]# mount -v -o remount /mnt/my_encrypted_backup/
mount: /dev/mapper/mybackup mounted on /mnt/my_encrypted_backup.
```

Running `lsblk` again produces the following output:

```
[root@rhel8 ~]# lsblk
NAME           MAJ:MIN RM  SIZE RO TYPE  MOUNTPOINT
sr0             11:0    1 1024M  0 rom
vda            252:0    0   30G  0 disk
├─vda1         252:1    0    1G  0 part  /boot
└─vda2         252:2    0   29G  0 part
  ├─rhel-root  253:0    0 26.9G  0 lvm   /
  └─rhel-swap  253:1    0  2.1G  0 lvm   [SWAP]
vdb            252:16   0   10G  0 disk
└─vdb1         252:17   0   10G  0 part
  └─mybackup   253:2    0   10G  0 crypt /mnt/my_encrypted_backup
```

Retrieving LUKS details

We can now dump the LUKS header information, data segment section, key slots used, etc.:

```
[root@rhel8 ~]# cryptsetup luksDump /dev/vdb1
LUKS header information
Version:        2
Epoch:          3
Metadata area:  12288 bytes
[……]
        Digest:     49 5a 68 e9 b6 66 50 2d c8 22 8e b9 d5 fd 2c af
                    23 b7 47 f3 2f 62 ee 6a b8 7c 93 8f 19 fe d8 3c
```

Adding a key file and automounting

Adding a key file and automounting

Mounting the LUKS encrypted filesystem automatically has security implications. For laptop users, doing this is not a wise choice. If your device gets stolen, so is your data that was stored in the encrypted partition.

Regardless of the security implication mentioned above, here's how to set up automatic mounting. First, create the appropriate directory to store the key file:

```
[root@rhel8 ~]# mkdir /etc/luks-keys/; dd if=/dev/random of=/etc/luks-keys/mybackup_key bs=32
count=1
[root@rhel8 ~]#
```

Then, add the key using the `cryptsetup` utility:

```
[root@rhel8 ~]# cryptsetup luksAddKey /dev/vdb1 /etc/luks-keys/mybackup_key
Enter any existing passphrase:
[root@rhel8 ~]#
```

Next, we need to restore the SELinux context:

```
[root@rhel8 ~]# restorecon -vvRF /etc/luks-keys

Relabeled /etc/luks-keys from unconfined_u:object_r:etc_t:s0 to system_u:object_r:etc_t:s0
Relabeled /etc/luks-keys/mybackup_key from unconfined_u:object_r:etc_t:s0 to
system_u:object_r:etc_t:s0
```

Previously, we opened the encrypted filesystem and mounted it manually. Now we need to see if we can do the same with automation. Since our filesystem is already mounted, we first need to `umount` (unmount) it:

```
[root@rhel8 ~]# umount /mnt/my_encrypted_backup
[root@rhel8 ~]# cryptsetup -v luksClose mybackup
Command successful.
```

Let's try opening the encrypted partition via the command line using the file as a key:

```
[root@rhel8 ~]# cryptsetup -v luksOpen /dev/vdb1 mybackup --key-file=/etc/luks-keys/mybackup_key

Key slot 1 unlocked.
Command successful.
```

Next, we need to configure `/etc/crypttab` and `/etc/fstab` to mount the disk on boot. We first need the UUID for `/dev/vdb1` (not `/dev/mapper/mybackup`), which can be retrieved as follows:

```
[root@rhel8 ~]# blkid /dev/vdb1

/dev/vdb1: UUID="46f89586-f802-44f1-aded-f80b16821189" TYPE="crypto_LUKS" PARTUUID="f92dbe33-01"
```

Now enter the following line in `/etc/crypttab` so we can automatically open our encrypted filesystem:

```
mybackup   UUID=46f89586-f802-44f1-aded-f80b16821189 /etc/luks-keys/mybackup_key luks
```

With this much done, we can now configure `/etc/fstab`. Append the following line (in bold) to this file:

```
[root@rhel8 ~]# vi /_etc_/fstab

#

# /etc/fstab
# Created by anaconda on Thu Aug  8 06:21:57 2019
#
# Accessible filesystems, by reference, are maintained under '/dev/disk/'.
# See man pages fstab(5), findfs(8), mount(8) and/or blkid(8) for more info.
#
# After editing this file, run 'systemctl daemon-reload' to update systemd
# units generated from this file.
#
/dev/mapper/rhel-root   /                       xfs     defaults        0 0
[...]
**/dev/mapper/mybackup /mnt/my_encrypted_backup xfs defaults 0 0**
```

And, finally, we can test to see if automount works without rebooting the machine, using `mount -a`:

```
[root@rhel8 ~]# mount -av

/                        : ignored
/boot                    : already mounted
swap                     : ignored
/mnt/my_encrypted_backup : successfully mounted
```

In this case, `/mnt/my_encrypted_backup` was successfully mounted. Now, reboot the system and make sure the automount works on reboot as well.

Final thoughts

There are other options that can be provided to `cryptsetup`, and each has trade-offs when it comes to speed and a more secure filesystem. Explore the options and choose what's best for your situation.

Topics: Linux Security

Valentin Bajrami

Valentin is a system engineer with more than six years of experience in networking, storage, high-performing clusters, and automation. He is involved in different open source projects like bash, Fedora, Ceph, FreeBSD and is a member of Red Hat Accelerators. More about me

Related Content