Hardening File Systems

Fortificación de S.O.
Master en Seguridad Informática. 2025/2026
Universidad da Coruña
Universidade de Vigo

Antonio Yáñez Izquierdo
José Rodríguez Pereira
David Otero

Contents I

Contents II

Basic concepts of files and directories in linux

Basic concepts of files and directories in linux $\rightarrow$files and directories

files and directories

- everything stored in the system is a file
  - a C program text
  - some letter
  - the executable file that constitutes a comand
  - the list of users in the system (`/etc/passwd`)
  - . . .
- security on unix depends greatly on access to files
- files are organized in a hierarchical structure. Looks like a *tree* but it is actually a graph
- this structure has a root directory designed by /

file permissions and ownership

- Each file in the system is owned by both ONE user and ONE group
  - the user owning the file may belong to several groups, but the file is owned only by one group
- The file has three sets of permissions associated (usually called the mode of the file)
- each set of permissions is a subset of the word **rwx**
  - the letter indicates the permission is granted
  - the - sign instead the letter indicates the permission is not granted

file permissions and ownership

- The first set are the permissions for the user owning the file, the second set the permissions for the group owning the file and the third set the permissions for the rest of the users in the system

r	the file can be read: view the file contents
w	the file can be written: modify the file contents, that is, the file can be appended, modified, overwritten ...
x	the file can be executed

file ownership and permissions

- example

  ```
  -rw-r----- 1 antonio audio 4656065 Sep 13 13:06 audiofile.mp3
  ```

- this file is owned by user `antonio` and group `audio`, its permissions are `rw-r-----` (the first - indicates it's a regular file)
  - the first set of permissions, `rw-`, means that a process from user antonio can read and write to the file
  - the second set of permissions, `r--`, means that a process from any user belonging to group audio can read the file
  - the third set of permissions, `---`, means that the rest of the users in the system can't read the file, nor write to it, neither execute it (were it an executable file)

permission representations

- the permissions are represented by an octal three digit number
  - one octal digit per set of permissions
  - to obtain the binary value for each of permissions we use 1 to represent the permission is granted and 0 otherwise
- that way `rw-r-----` is represented as binary 110 100 000 and octal 640
- `rwxr-xr--` would be represented as binary 111 101 100 and octal 754

special permissions

- there are three more special permissions
    - **sticky bit** (octal 1000) (`--------t`). On modern systems it has no effect on files, on older systems, if one executable had the sticky bit set, the system would not deassign the swap space after executing the file
    - **setgid** (octal 2000) (`-----s---`) The process excuting a file with the setgid bit set gets the group credential of the group owning the executable
    - **setuid** (octal 4000) (`--s------`) The process excuting a file with the setuid bit set gets the user credential of the user owning the executable

example of special permissions

- Consider the file
  ```
  -rwsr-sr-x 1 antonio audio 4656065 Sep 13 13:06 program1.out
  ```
- its permissions are rwsr-sr-x (binary 110 111 101 101, octal 6755)
- processes from user antonio can read write and execute the file
- processes from users belonging to group audio can read and execute the file
- processes from any user can read and execute the file
- a process executing the file gets its user credential changed to that of user *antonio* and its group credential changed to that of group *audio*

permissions in directories

- the permissions in directories have the following meaning
  - **r** The directory can be listed: see the names of the files in it
  - **w** The contents of the directory can be modified: files can be added to it or files can be removed from it
  - **x** The files in the directory can be accessed
  - **setgid** Files created in this directory get owned by directory group instead of the group of the process creating the file
  - **sticky bit** Only the owner (or one who has write acess) of a file can delete it, even having write access to the directory

Basic concepts of files and directories in linux $\rightarrow$other types of files

other types of files

- In addition to files and directories unix supports the following types of files
  - **block devices**. They have no assigned space, just two numbers (major a minor) used to tell the kernel which device driver to use when accessing the device
  - **character devices**
  - **symbolic links** A file whose *contents* are the path to the file the link refers to
  - **fifo** A first in first out file

other types of files

- non symbolic links are not special files, they are just another name to an existing file
- the comand `ls -l` lets us distinguish the different types of files in a unix system

```
abyecto:/home/antonio/pru# ls -l
total 12
brw-r--r-- 1 root root 15,  3 Sep 13 18:02 block_device
crw-r--r-- 1 root root  9, 51 Sep 13 18:14 char_device
-rw-r--r-- 2 root root     93 Sep 13 18:03 file
-rw-r--r-- 2 root root     93 Sep 13 18:03 link
lrwxrwxrwx 1 root root      4 Sep 13 18:18 symlink -> file
drwxr-xr-x 2 root root   4096 Sep 13 18:01 this_is_a_directory
prw-r--r-- 1 root root      0 Sep 13 18:03 this_is_a_fifo
```

Basic concepts of files and directories in linux
→commands for dealing with files

usual commands to access files

- these are the most usual commands to access files in a unix system. The online *man* page is the ultimate source of information in the system we're using

`mv`	moves (or renames) a file or directory
`cp`	copies files or directories
`chown`	changes the owner of a file (must be root)
`chmod`	changes the mode (permissions) of a file (must be owner)
`chgrp`	changes the group of a file (must be owner)
`mkdir`	creates a directory

usual commands to access files

`mknod`	creates a special file (directory, device or fifo)
`mkfifo`	creates a special fifo file
`ln`	creates a link (both symbolic and non symbolic)
`rm`	removes a file
`rmdir`	removes a directory
`ls`	lists the contents of a directory
`cd`	changes directory
`umask`	sets the file creation mask (default permissions)

Introduction to filesystems in linux

Introduction to filesystems in linux
→Discs, partitions and filesystems

Discs and partitions

- in linux we create filesystems on block devices to store our information in files
- block devices are named `/dev/sda`, `/dev/sdb`, `/dev/sdc` . . ., this includes disks with different interfaces (usb, SATA, SCSI, IDE, usb pendrives, memory cards . . .) although in older linux kernels IDE hard drivers were named ... /dev/hda, /dev/hdc. . .
- this devices are usually *partitioned* and the partitions in them named `/dev/sda1`, `/dev/sda2`, `/dev/sda3` . . . . Most common partition formats in the linux world are the MBR and GPT formats (as seen on the lesson about hardening boot)
- progams to partition disk are `fdisk`, `cfdisk`, `parted`, `gpart` . . .

Discs and partitions

- older devices, such as floppy discs (typically named /dev/fd0, /dev/fd1 . . .) were used without partitioning.
- Sometimes usb pendrives and memory cards are used without partitioning, although it is not the usual method
  - we would use '`mkfs /dev/sdc`' to create a file system on an usb drive (were its device sdc)
  - we would use '`mkfs /dev/sdc1`' to create a file system on the first patition of an usb drive (were its device sdc)

Creating filesystems

- prior to accessing filesystems they must be created (the equivalent to *windows formatting*)
- filesystems can be created with the command `mkfs`
  mkfs -t type [fs-options] device
- `mkfs` is just a frontend, when we call `mkfs -t ext4` we are actually calling the command `mkfs.ext4`. The latter is the preferred method
- in the following lines we create an *ext2* filesystem with a blocksize of 4K in `/dev/sda4` and a filesystem of type *ext4* with an inodesize of 256 bytes on `/dev/mapper/lvm0`

```
# mkfs -t ext2 -b 4096 /dev/sda4
.....
# mkfs.ext4  -I 256 /dev/mapper/lvm0
```

Accessing filesystems

- to access a filesystem we have to *mount* it at some direcory. The filesystem then becommes accesible under that directory (in windows, each filesystem is assigned a drive letter, for example `C:`)
- linux understands different type of filesystems: `ext2`, `ext3`, `ext4`, `jfs`, `vfat`, `ntfs`, `udf`, `iso9660` `...`, so the syntax of the mount command is

  ```
  mount -t filesystemtype -o comma-separated-options device directory
  ```

  The foolowing example mounts the filesystem of type ext4 in `/dev/sdb5` onto directory `/mnt` in a readonly and noexec mode

  ```
  # mount -t ext4 -o ro,noexec /dev/sdb5 /mnt
  ```

Accessing filesystems

- the root filesystem is specified via the boot loader.
- Filesystems mounted automatically at boot type (also including the root filesystem), are specified in the `/etc/fstab` file
- the format of the `/etc/fstab` is simple, one line per file system to mount, with fields separated by blanks

format of the `/etc/fstab` file

- each line of the `/etc/fstab` file has the following fields

device	device to mount
dir	directory to mount onto
type	type of filesystem
opts	comma separated list of mount options (filesystem type dependant)
dump	1 or 0 depending whether the filesystem backup is controlled by the *dump* command
pass	specifies if the device is checked at boot time

format of the /etc/fstab file

- the most usual mount options are *defaults*, *ro*, *rw*, *nosuid*, *nexec*, *noauto*, *usrquota* . . .
- example of /etc/fstab file

```
 # <file system> <mount point> <type> <options> <dump> <pass>
/dev/sda1          /           ext4    defaults      0      1
/dev/sda7          none        swap      sw          0      0
/dev/sda5          /var        xfs     noexec,nosuid 0      1
/dev/cdrom         /cdrom      iso9660 defaults,ro,user,noauto 0   0
```

format of the `/etc/fstab` file

- in modern linux systems, we can, instead of using the name of the device (i.e. /dev/sda4), use the UUID (Universally Unique IDentifier)
- so, an entry in the `/etc/fstab` file will look like this

```
UUID="d39577d4-fb9b-4b18-be4e-53ff32dbf856" /home ext4 noatime  0   2
```

- that number can be obtained with the command **blkid**. Example

```
root@abyecto:/home/antonio# blkid /dev/sdb2
/dev/sdb2: UUID="7b127a41-0ff1-45ed-8c0a-dac6816cd02c" TYPE="ext2" PARTUUID="e9ddd7cb-911f-3b47-916b-d7c9
root@abyecto:/home/antonio#
```

Partitions and Logical Volumes

- there are two approaches to using discs and partitions in linux
  1. we create filesystems on devices such as discs or partitions and we mount them onto directories. This is the traditional approach
  2. we combine different devices to create a Logical Volume, and we create the filesystem on top of the Logical Volume. Space can be added to the logical volume at a latter time
- either case, the filesystems need to be mounted to be accessed

Introduction to filesystems in linux
→Filesystems in partitions

Filesystems in partitions

- this is the traditional simple approach
- we create a file system onto each physical device (disc or partition)
- that filesystem must be mounted to be accessed
- as partitions cannot be easily resized this way of doing things lacks flexibility
  - (the program `resizefs` can shrink a filesystem or enlarge it, provided there's room at the end of the partition)
- unless we explicitly select LVM during installation, well get this approach. It is also the preferred method for removable media
- this type of file systems can be crypted

Introduction to filesystems in linux →LVM

lvm

- Logical volumes are more flexible than the traditional approach as we can dynamically add space to a filesystem
- naming schemes can be more clear and thus easier to administer
- Logical Volumes can be crypted
- not advisable for the `/boot` partition
- to use LVM in debian type distros the `lvm2` package must be installed

LVM: Physical Volumes

- a Physical Volume is either a disk, or a partition (or even a file using the loopback device) configured as such
- it consists of a header and a certain number of physical extents (which are the smallest contiguous extent in the physical volume that can be assigned to a Logical Volume). The default size for a logical extent is 4 Mb
- to create a Physical Volume on a partition
  - we create a partition (MBR or GPT) with the appropiate tag code (0x8e, 0x8e00: Linux LVM)
  - we create the Physical Volume on it (which basically creates the header) with the command `pvcreate`

    ```
    #pvcreate /dev/sda2
    ```

  - we can track created PVs with the command `pvdisplay`

LVM: Volume Group

- a Volume Group is a collection of PV grouped together
- we can create one with the command **vgcreate** as in the following example (sda4, sdb2 and sdc1 are suposed to be PVs already created)

```
# vgreate NuevoVol /dev/sda4 /dev/sdb2 /dev/sdc1
```

- or we can first create it on a PV and the extend it

```
# vgreate NuevoVol /dev/sda4
# vgextend NuevoVol /dev/sdb2
# vgextend NuevoVol /dev/sdc1
```

LVM: Logical Volume

- we create Logical Volumes on top of the Volume Group giving then a name and specifying the size of the LV. More than one LV can be created on a Volume Group

  ```
  # lvcreate -L 2G NuevoVol -n LVDatos
  ```

  creates a Logical Volume of 2G named *LVDatos* on Volume Group *NuevoVol*
- the command `lvdisplay` shows it, `lvresize` allows us to change its size . . .
- we now have the special file `/dev/NuevoVol/LVDatos` where we can create a filesystem on to be mounted afterwards
- we can also create a line in the `/etc/fstab` file to get the Logical Volume mounted at boot time

LVM: Logical Volume

- output of the command lvdisplay

```
root@hardeningB:/home/antonio# lvdisplay
 --- Logical volume ---
 LV Path                /dev/NuevoVol/LVDatos
 LV Name                LVDatos
 VG Name                NuevoVol
 LV UUID                VpThC4-ARJo-ucQ6-7EUz-1BYX-8X1V-uYGp1m
 LV Write Access        read/write
 LV Creation host, time hardeningB, 2019-01-31 20:36:07 +0100
 LV Status              available
 # open                 0
 LV Size                2.00 GiB
 Current LE             512
 Segments               1
 Allocation             inherit
 Read ahead sectors     auto
 - currently set to     256
 Block device           254:0
```

LVM: Logical Volume

- a Logical Volume can be easily resized, even if mounted

```
root@hardeningB:/home/antonio# df /datos/
Filesystem                     1K-blocks  Used Available Use% Mounted on
/dev/mapper/NuevoVol-LVDatos   3030800  6144   2850988   1% /datos
root@hardeningB:/home/antonio# lvresize -L 4G -r NuevoVol/LVDatos
  Size of logical volume NuevoVol/LVDatos changed from 3.00 GiB (768 extents) to 4.00 GiB (1024 ext
  Logical volume NuevoVol/LVDatos successfully resized.
resize2fs 1.43.4 (31-Jan-2017)
Filesystem at /dev/mapper/NuevoVol-LVDatos is mounted on /datos; on-line resizing required
old_desc_blocks = 1, new_desc_blocks = 1
The filesystem on /dev/mapper/NuevoVol-LVDatos is now 1048576 (4k) blocks long.

root@hardeningB:/home/antonio# df /datos/
Filesystem                     1K-blocks  Used Available Use% Mounted on
/dev/mapper/NuevoVol-LVDatos   4062912  8184   3839120   1% /datos
```

- We see that we have mounted `/dev/NuevoVol/LVDatos` and that the device that appears actually mounted is `/dev/mapper/NuevoVol-LVDatos`. They are both symlinks to the actual device used (probably `/dev/mapper/dm-0`)

Possible threats

Posible threats to filesystems

- the main threats to a file system are
  a) unathorized access to information stored on it
  b) filling the file system thus preventing other users to write to the file system
  c) corrupting the file system making it unusable
  d) (on networked or removable fylesystems) gaining access to malicious executable files with increased privileges

Posible threats to filesystems

- as far as unathorized access goes, we have to be carefull of file premissions. Some tips are
  - all home directories must be 700. We can set *umask* to be 077
  - configuration files for different daemons need not be world-readable
  - take away execution permissions not strictly needed
  - all system's directories must be non-writable execpt for the system administrator *root*
  - `/tmp` must be world writable but with the sticky bit on *rwxrwxrwt*

Posible threats to filesystems

- some linux distros have programs that check (and set) all the file permissions in the filesysem on a periodic basis (for example, Mandriva's `msec`)
- if we feel that the approach owner-group-world is not enough and we need to finetune the file permissions we can make use of **ACLs**

Posible threats to filesystems

- we can also *crypt* our filesytems to prevent unathorized access from someone gaining physical access to the machine
- to prevent a user (or a group of users, for that matter) from filling up the file system we can make use of **quotas**
- the **nosuid** mount option should be always used on networked and removable file systems. On removable file systems the option **noexec** is also advisable

Posible threats to filesystems

- to prevent file system corruption we must
  - use a tested and reliable filesystem, avoiding experimental features
  - keep the kernel patched version and do not use an experimental filesystem layer
  - watch carefully for device file permissions (files in the `/dev` directory)
  - keep the machine in an stable environment (constant power supply, adecuate temperature . . .)
  - physically protect the machine

ACLs

ACLs

- ACLs stands for Access Control Lists
- it's a mechanism that allows us to define file permissions explicitly for each user and/or group (not just owner-group-world approach)
- linux implements POSIX ACLs
- ACLs can be stablished with the command **setfacl**
- ACLS can be checked with the command **getfacl**

ACLs

- to use ACLs we have to have ACL support in the kernel (if we use a standard distribution kernel we most likely have, if we compiled it ourselves...we know)
- the filesystems were we want to have ACLs must be mounted with the option `acl`
  - we can specify that option in the `/etc/fstab` file
  - we can check the mount options in `/proc/mounts`
- to set specific user and group permissions on a file we use the following commands

```
# setfacl -m "u:user:permissions" <file/dir>

# setfacl -m "g:group:permissions" <file/dir>
```

ACLs: example

- the following example sets read and write permissions for user antonio on file *nuevofich* which is root owned and has permissions 600
- after that, user antonio is allowed to read and write to the file
- note that before changing the file's ACL `ls -l` showed the standard permissions `rw-------`
- after changing the ACL `ls -l` showed `rw-------+` with a + sign, indicating that the file has ACL

ACLs: example

```
root@hardeningB:/datos# ls -l nuevofich
-rw------- 1 root root 0 Feb  1 11:11 nuevofich
root@hardeningB:/datos# getfacl nuevofich
# file: nuevofich
# owner: root
# group: root
user::rw-
group::---
other::---
root@hardeningB:/datos# setfacl -m u:antonio:rw nuevofich
root@hardeningB:/datos# ls -l nuevofich
-rw-rw----+ 1 root root 0 Feb  1 11:11 nuevofich
root@hardeningB:/datos# getfacl nuevofich
# file: nuevofich
# owner: root
# group: root
user::rw-
user:antonio:rw-
group::---
mask::rw-
other::---
root@hardeningB:/datos#
```

Quotas

Quotas

- *quotas* allow us to restrict the amount of space a user (or a group) can use on a file system
- *quotas* configuration is per user (or group) and filesystem
- *quotas* reside in the files `aquota.user` (or `aquota.group` in the root directory of the filesystem where the quota is established

Quotas

- for each user (or group) in a file system we can establish a limit both on the files (*inodes*) and blocks (*space*) that a user or group can use. This is what we call the quota
- for each user (or group) in a file system both a soft and a hard limit (for both files and blocks) are configured
  - upon reaching the soft limit a warning is issued, but the write system calls still work
  - upon reaching the hard limit write system calls fail, so the user (or group) can never exceed the hard limit
  - the user (or group) can stay over the soft limit for a period of time (call *grace period*) after which the soft limit becomes the hard limit (write system calls fail)

Enabling quotas

- to enable *quotas* on a file system we need
  1. have *quota* support in the kernel (mostly all preconfigured distro kernels come with quota support)
  2. mount the file system with the *usrquota* (and/or *grpquota*) option
  3. have installed the corresponding *quota* management programs (in debian type distros `apt-get install quota`)

Enabling quotas: utilities

- the *quota* package includes the following utilities
  - **quotacheck** creates, checks and/or repairs quota files in a files system
  - **quotaon, quotaoff** turns on (or off) quotas on a files system
  - **edquota** allows modification of a user (or group) quotas
  - **repquota, quota** reports the status of the quotas in a file system

Defining quotas: edquota

- we use the program **edquota** to establish quotas for different users. A summary of its usage is
  - **edquota -u name** opens the editor defined in $EDITOR for us to modify the soft and hard limits for user *name*.
  - **edquota -g grpname** opens the editor defined in $EDITOR for us to modify the soft and hard limits for group *grpname*.
  - **edquota -p prototype name** establishes quotas for user *name* the same as user *prototype*.
  - **edquota -t** establishes the grace period

Defining and stablishing quotas: example

- in the following example
  - we'll create the quota files (for both user and group) in the filesystem at `/dev/NuevoVol/LVDatos`
  - we'll establish quotas in the filesystem at `/dev/NuevoVol/LVDatos` for user *antonio* and group *bin*
  - we'll turn on quotas for that file system
  - we'll make every user defined locally in the system with the /bin/bash as his/her login shell have the same quota as user *antonio*
- note that the next time we boot the system, if the filesystem is mounted with the quota options on `/etc/fstab` the booting scripts will take care of checking and turning the quotas on, so we need do nothing

Defining and stablishing quotas: example

```
.
root@hardeningB:/home/antonio#
root@hardeningB:/home/antonio# mount -t ext4 -o usrquota,grpquota /dev/NuevoVol/LVDatos  /datos/
root@hardeningB:/home/antonio#
root@hardeningB:/home/antonio#
root@hardeningB:/home/antonio# quotacheck -uv /datos/
quotacheck: Your kernel probably supports journaled quota but you are not using it. Consider switching to
quotacheck: Scanning /dev/mapper/NuevoVol-LVDatos [/datos] done
.....
quotacheck: Old file not found.
root@hardeningB:/home/antonio#
root@hardeningB:/home/antonio# quotacheck -gv /datos/
quotacheck: Your kernel probably supports journaled quota but you are not using it. Consider switching to
quotacheck: Scanning /dev/mapper/NuevoVol-LVDatos [/datos] done
quotacheck: Checked 2 directories and 2 files
quotacheck: Old file not found.
root@hardeningB:/home/antonio#
root@hardeningB:/home/antonio# quotaon /dev/NuevoVol/LVDatos
root@hardeningB:/home/antonio#
root@hardeningB:/home/antonio# edquota -u antonio
root@hardeningB:/home/antonio# edquota -g bin
root@hardeningB:/home/antonio# edquota -t
root@hardeningB:/home/antonio# for name in ‘cat /etc/passwd | grep /bin/bash | cut -f1 -d:‘ ;
do  edquota -p antonio $name ; done
```

Quotas
→Quotas on ext4 filesystems

Quotas on ext4 filesystems

- Quotas reside on the root directory of the filesystem they define (files `aquota.user` and `aquota.group`)
- Ext4 filesystems also support quotas as extended attributes. Modern `quotacheck` utilities refuse to update (create) the quota files if they do not already exist on an ext4 filesystem. We must used the `-c` (create) option and `-m` (advisable: do not mount read-only) in `quotacheck` to create the quota files.
    - Example, to create quota files on /dev/sda7
      `quotacheck -ucmv /dev/sda7`

Quotas on ext4 filesystems

- In addition to the previous method, we can also create quotas on a ext4 filesystems the *"ext4 way"*. The following example is for user quotas.
- Create the filesystem with the quota option and the adecuate quotatype as attribute. (should the filesystem be already created we can use `tune2fs` to change the options)

```
# mkfs.ext4 -O quota -E quotatype=usrquota  /dev/sda6
```

- Mount the filesystem with the adecuate quota option

```
# mount -o usrquota /dev/sda6 /mnt
```

- Activate the quota with `quotaon`

```
# quotaon /dev/sda6
```

- And finally, edit the user quotas with `edquota -u`

Crypting

Crypting
→Crypting partitions

Crypting partitions

- we use the module dm-crypt, which creates a link between `/dev/mapper/our-chosen-name` and the real crypted device (for example `/dev/sdb4`)
- we deal with an uncrypted device `/dev/mapper/our-chosen-name` and it is transparently crypted to the real device (for example `/dev/sdb4`) with the key that remains in kernel memory
- we supply the software with a *passphrase* with which it generates the real key to crypt. The passphrase can also be read from a file `--key-file` option
  - not a good idea to use non ASCII chars in the passphase
- the hash used to create the key from the passphrase is configurable, as is the cipher algorithm.

Crypting partitions

- we have two ways of using `cryptsetup` (frontend to dm-crypt)
- **plain type**
  - the link is a link between the crypted and the plain device; the encryption options to be employed are used directly to create the mapping between an encrypted disk and a named device
  - we can create the mapping against a partition or a full device. In the latter we can get a crypted partition table o no partitions at all
- **LUKS type**
  - LUKS creathes aheader on the device, with the crypting options and the masterkey crypted using the passphrase
  - it is compatible with some windows software
  - the passphrase can be changed if we have root access to the volume
- all the following examples use a keyboard entered passphrase and the default options

Crypting partitions: plain type example

- we create the crypting device

```
root@hardeningB:/home/antonio#cryptsetup open /dev/sdb4 cifrado --type plain
Enter passphrase:
root@hardeningB:/home/antonio#
```

- we make the filesystem on it and mount it

```
root@hardeningB:/home/antonio# mkfs.ext4 /dev/mapper/cifrado
mke2fs 1.43.4 (31-Jan-2017)
Creating filesystem with 786176 4k blocks and 196608 inodes
Filesystem UUID: d0492b11-bff6-40d3-8bc5-d957b1b6880d
Superblock backups stored on blocks:
 32768, 98304, 163840, 229376, 294912

Allocating group tables: done
Writing inode tables: done
Creating journal (16384 blocks): done
Writing superblocks and filesystem accounting information: done

root@hardeningB:/home/antonio# mount /dev/mapper/cifrado /mnt/
```

- from that point on, we can use the file system. When we're done we can umount it and remove the crypting device

```
root@hardeningB:/home/antonio# umount /mnt/
root@hardeningB:/home/antonio# cryptsetup close cifrado
root@hardeningB:/home/antonio#
```

Crypting partitions: plain type example

- if we want to use the crypted file system, we only need to do

```
root@hardeningB:/home/antonio#cryptsetup open /dev/sdb4 cifrado --type plain
Enter passphrase:
root@hardeningB:/home/antonio# mount /dev/mapper/cifrado /mnt/
```

- Now we can use it. After that we can umount it and remove the crypting device

```
root@hardeningB:/home/antonio# umount /mnt/
root@hardeningB:/home/antonio# cryptsetup close cifrado
root@hardeningB:/home/antonio#
```

- note that we can not change the passpharase and as we did not specify options for cryptsetup we are using the defaults, (*ripemd160* to create key from passpharese and *aes-cbc-essiv:sha256* cipher) which we should remember because they are stored nowhere

Crypting partitions: LUKS type example

- first we write the LUKS header on the device, the **-y** parameter is to be asked for the passphrase twice

```
root@hardeningB:/home/antonio# cryptsetup -y -v  luksFormat --type luks /dev/sdb4

WARNING!
========
This will overwrite data on /dev/sdb4 irrevocably.

Are you sure? (Type uppercase yes): YES
Enter passphrase:
Verify passphrase:
Command successful.
root@hardeningB:/home/antonio#
```

- we now open the crypting device. Note that if we misstype the passphrase we get an error that no key associated with that passphrase

```
 root@hardeningB:/home/antonio# cryptsetup open /dev/sdb4 encriptadoLUKS
Enter passphrase for /dev/sdb4:
No key available with this passphrase.
Enter passphrase for /dev/sdb4:
root@hardeningB:/home/antonio#
```

Crypting partitions: LUKS type example

- now we can proceed to create the filesystem, mount it , use it, and umount it and close the crypting device when we are done

```
root@hardeningB:/home/antonio# mkfs.ext4 /dev/mapper/encriptadoLUKS
mke2fs 1.43.4 (31-Jan-2017)
Creating filesystem with 785664 4k blocks and 196608 inodes
Filesystem UUID: 584e9808-2334-4367-b70f-e3fdad5c8187
Superblock backups stored on blocks:
 32768, 98304, 163840, 229376, 294912

Allocating group tables: done
Writing inode tables: done
Creating journal (16384 blocks): done
Writing superblocks and filesystem accounting information: done
root@hardeningB:/home/antonio# mount /dev/mapper/encriptadoLUKS /mnt
......
root@hardeningB:/home/antonio# umount /mnt/
root@hardeningB:/home/antonio# cryptsetup close encriptadoLUKS
root@hardeningB:/home/antonio#
```

- should we want to change the passphrase

```
root@hardeningB:/home/antonio# cryptsetup luksChangeKey /dev/sdb4
Enter passphrase to be changed:
Enter new passphrase:
Verify passphrase:
root@hardeningB:/home/antonio#
```

Crypting partitions: several passphrases

- LUKS allows us to store use up to eight passphrases for the same device
- we can add, revoke and change passphrases on an individual basis
- in the example we add a passphrase to device /dev/sdb4

```
oot@hardeningBi:~# cryptsetup luksAddKey /dev/sdb4
Enter any existing passphrase:
Enter new passphrase for key slot:
Verify passphrase:
```

Crypting
→Crypting LVMs

Crypting LVMS

- there are several ways to tho this, although the simplest are
  - build the Physical Volumes on plain crypted devices
  - build the Physical Volumes on LUKS crypted devives
- in the following example we are going to build two Logical Volumes (Datos1 and Datos2) on a VG with one LUKS crypted PV.Then we open the crypted device
- first we create the LUKS header

```
root@hardeningB:/home/antonio# cryptsetup luksFormat --type luks /dev/sdb4

WARNING!
========
This will overwrite data on /dev/sdb4 irrevocably.

Are you sure? (Type uppercase yes): YES
Enter passphrase:
Verify passphrase:
root@hardeningB:/home/antonio# cryptsetup open /dev/sdb4 PVcriptao
Enter passphrase for /dev/sdb4:
root@hardeningB:/home/antonio#
```

Crypting LVMS

- then we create a Physical Volume on it and check its ok

```
  root@hardeningB:/home/antonio# pvcreate /dev/mapper/PVcriptao
  Physical volume "/dev/mapper/PVcriptao" successfully created.
root@hardeningB:/home/antonio# pvdisplay
  --- Physical volume ---
  PV Name               /dev/sdb1
  .............................(this were created before)
  Free PE               254
  Allocated PE          257
  PV UUID               cGVydZ-A7Mw-up38-yC1x-MCFH-cLk3-7SzPzY

  "/dev/mapper/PVcriptao" is a new physical volume of "3.00 GiB"
  --- NEW Physical volume ---
  PV Name               /dev/mapper/PVcriptao
  VG Name
  PV Size               3.00 GiB
  Allocatable           NO
  PE Size               0
  Total PE              0
  Free PE               0
  Allocated PE          0
  PV UUID               Kc2XdS-32WF-purA-FSA0-mpMp-Ff0X-0Kn3W4

root@hardeningB:/home/antonio#
```

Crypting LVMS

- then we create a Volume Group on it and check it's ok

```
 root@hardeningB:/home/antonio# vgcreate VolumeCriptao /dev/mapper/PVcriptao
  Volume group "VolumeCriptao" successfully created
root@hardeningB:/home/antonio# vgdisplay
  --- Volume group ---
  VG Name               NuevoVol
........................................(the one created before)

  --- Volume group ---
  VG Name               VolumeCriptao
  System ID
  Format                lvm2
  Metadata Areas        1
  Metadata Sequence No  1
  VG Access             read/write
  VG Status             resizable
  MAX LV                0
  Cur LV                0
  Open LV               0
  Max PV                0
  Cur PV                1
  Act PV                1
  VG Size               3.00 GiB
  PE Size               4.00 MiB
  Total PE              767
  Alloc PE / Size       0 / 0
  Free  PE / Size       767 / 3.00 GiB
  VG UUID               DR5yYQ-f7C5-U4Ml-hT25-Vy0t-QW1F-yxJC0z

root@hardeningB:/home/antonio#
```

Crypting LVMS

- finally we create the two Logical Volumes where we can make filesystems and then have them mounted

```
root@hardeningB:/home/antonio# lvcreate -L 1G VolumeCriptao -n Datos1
  Logical volume "Datos1" created.
root@hardeningB:/home/antonio# lvcreate -L 1G VolumeCriptao -n Datos2
  Logical volume "Datos2" created.
root@hardeningB:/home/antonio# mkfs.ext4 /dev/VolumeCriptao/Datos1
mke2fs 1.43.4 (31-Jan-2017)
Creating filesystem with 262144 4k blocks and 65536 inodes
Filesystem UUID: 8aefbaa6-5e66-40de-a308-2974fdd5a4db
Superblock backups stored on blocks:
 32768, 98304, 163840, 229376

Allocating group tables: done
Writing inode tables: done
Creating journal (8192 blocks): done
Writing superblocks and filesystem accounting information: done

root@hardeningB:/home/antonio# mkfs.ext4 /dev/VolumeCriptao/Datos2
mke2fs 1.43.4 (31-Jan-2017)
Creating filesystem with 262144 4k blocks and 65536 inodes
.......................................................
Creating journal (8192 blocks): done
Writing superblocks and filesystem accounting information: done

root@hardeningB:/home/antonio#
```

Crypting
→encfs

Crypting partitions and LVMs overhead

- crypting partitions (and logical volumes for that matter) is a system-wide solution
- must be *root* to do that
- the passphrase must be input (either by keyboard or removable media) at boot (which might be an inconvenience)
- we end up crypting much information unnecessarily (mostly system files)
- sometimes we need a more simple solution
  - crypting only the home directories needed (as done by ubuntu, crypted directories reside in files and are decrypted and mounted upon login)
  - let the user be able to crypt the directories he feels like without bothering the system administrator (this is what **encfs** does)

Crypting directories with encfs

- we only need to install the package **encfs** (in debian type distros `apt-get install encfs`)
- Now we create the crypted directory onto a clear one

```
antonio@hardeningB:~$ encfs /home/antonio/Crypted/ /home/antonio/Clear/
Creating new encrypted volume.
Please choose from one of the following options:
 enter "x" for expert configuration mode,
 enter "p" for pre-configured paranoia mode,
 anything else, or an empty line will select standard mode.
?>

Standard configuration selected.
Configuration finished.  The filesystem to be created has
the following properties:
Filesystem cipher: "ssl/aes", version 3:0:2
Filename encoding: "nameio/block", version 4:0:2
Key Size: 192 bits
Block Size: 1024 bytes
Each file contains 8 byte header with unique IV data.
Filenames encoded using IV chaining mode.
File holes passed through to ciphertext.

Now you will need to enter a password for your filesystem.
You will need to remember this password, as there is absolutely
no recovery mechanism.  However, the password can be changed
later using encfsctl.

New Encfs Password:
```

Crypting directories with encfs

- in this case our decrypted directory is /home/antonio/Clear and the crypted (real) files reside in /home/antonio/Crypted

```
antonio@hardeningB:~$ ls Clear
Desktop  Documents  Downloads
antonio@hardeningB:~$ ls Crypted/
dPs3OXDuMOBNoY-E3xxer1kQ  jHmBig77BxJ5p6WRFoqrRlTe  T4siEtLQ,9jkWBRDsTzjE2w0
antonio@hardeningB:~$
```

- to umount the directory Clear (so that we lose access to the files)

```
 antonio@hardeningB:~$ fusermount -u /home/antonio/Clear
```

- to have it mounted again

```
 antonio@hardeningB:~$ encfs /home/antonio/Crypted/ /home/antonio/Clear/
```

(and supply the passphrase)

Crypting directories with encfs

- passphrase can be changed with the **encfsctl** command
- important information resides in the file /home/antonio/Crypted/.encfs6.xml including the cypher algorithm and the (crypted form) key
    - loss of this file means loss of the whole contents of the crypted directory

Locking directories and restricting access to devices

Other considerations

- There might be also some interest in locking down some system directories: a perfect candidate is the `/boot` directory, which contains files needed to boot the system
  - as this directory does usually reside in an independent partition, we can consider having it mounted read-only, the downside is that this way we cannot perform a kernel upgrade unless we remount it read-write. As a bonus, this prevents the boot configuration from being changed
- we might also want to restrict access to some devices, an interesting way of doing this is blacklisting the kernel modules that give support to that devices.
  - we can disable access to usb by adding `'blacklist usb_storage'` to file `/etc/modprobe.d/blacklist.conf`