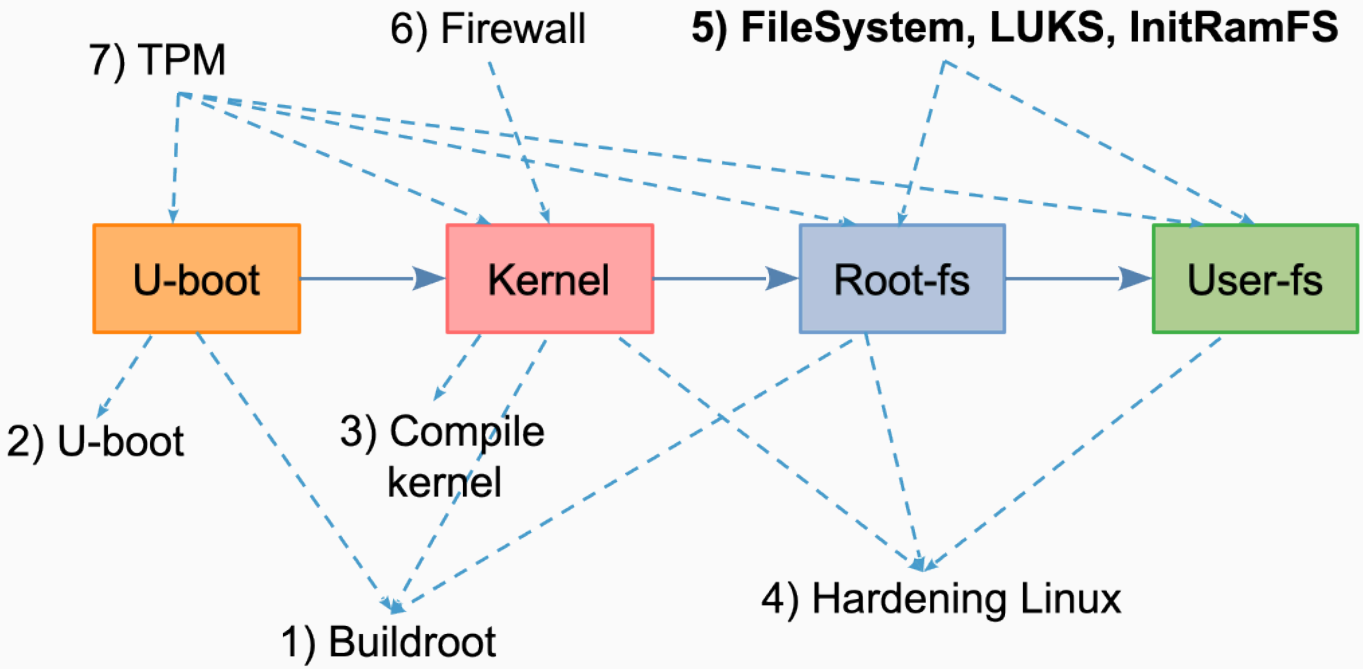
MSE | MASTER OF SCIENCE IN ENGINEERING

MA_SeS - File system Security

Luca Haab, Michael Mäder, Florent Glück

November 02, 2025

Overview


7) TPM
6) Firewall
5) FileSystem, LUKS, InitRamFS
U-boot
Kernel
Root-fs
User-fs
2) U-boot
3) Compile kernel
4) Hardening Linux
1) Buildroot

References

[1] Real World, Linux Security, Bob Toxen, 0-13-046456-2

[2] Hacking Exposed Linux, Third Edition, Isecom, 978-0-07-226257-5

[3] http://computingtech.blogspot.com/2010/02/embedded-linux-security.html

[4] Advanced UNIX Programming with Linux, Mark Mitchell, Jeffrey Oldham, and Alex Samuel, www.newriders.com, 0-7357-1043-0

[5] www.cyberciti.biz/tips/

[6] www.netzmafia.de/skripten/unix/linux-daemon-howto.html

[7] en.kioskea.net/faq/1170-linux-managing-file-attributes-on-ext2

[8] www.kernel.org/pub/linux/libs/pam

[9] www.techrepublic.com/blog/opensource/secure-temporary-files-in-linux/171

[10] publib.boulder.ibm.com/infocenter/aix/v6r1/index.jsp?topic=%2Fcom.ibm.aix.files%2Fdoc%2Faixfiles%2Finittab.htm

[11] g0tmi1k.blogspot.com/2011/08/basic-linux-privilege-escalation.html?m=1

[12] http://www.siteduzero.com/informatique/tutoriels/les-acl-access-control-lists-sous-linux

[13] www.yolinux.com/TUTORIALS/LinuxTutorialManagingGroups.html

inode

An **inode** (Index Node) is a data structure in Unix/Linux filesystems that stores metadata about a file or directory.

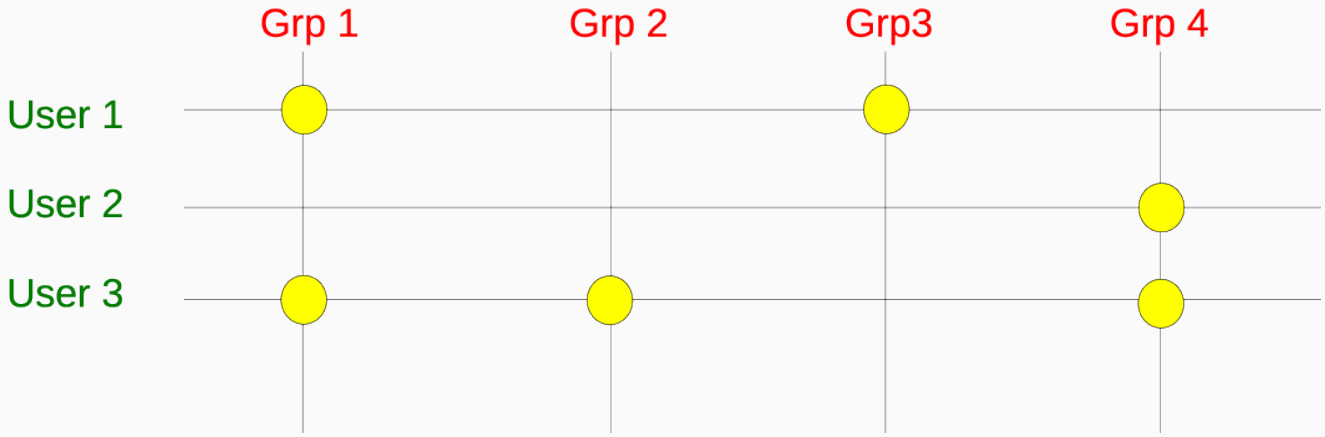
Each file has an inode. Each inode stores the attributes and disk block locations of the object's data

Mode	[0100644]
User ID	[0]
Group ID	[0]
Size	[7]
Creation time	[1690953428]
Modification time	[1690953428]
Access time	[1690958443]
Deletion time	[0]
Link count	[1]
Block count high	[0]
Block count	[8]
File flags	[0x0]
Generation	[0x83bbaed2]
File acl	[0]
High 32bits of size	[0]
Fragment address	[0]
Direct Block #0	[512]
Direct Block #1	[0]
Direct Block #2	[0]
Direct Block #3	[0]
Direct Block #4	[0]
Direct Block #5	[0]
Direct Block #6	[0]
Direct Block #7	[0]
Direct Block #8	[0]
Direct Block #9	[0]
Direct Block #10	[0]
Direct Block #11	[0]
Indirect Block	[0]
Double Indirect Block	[0]
Triple Indirect Block	[0]

Users and Groups

User and Groups (1/4)

- **User**: Each user is assigned a unique number, called a user ID, or UID.
- **Group**: Each group is assigned a unique number, called a group ID, or GID. Every group contains one or more user IDs. A single user ID can be a member of some of groups, but groups can't contain other groups; they can contain only users. Like users, groups have names.



User and Groups (2/4)

- `/etc/passwd` contains the user list
- `/etc/group` contains the group list and the user list for each group

```
# ls -al /etc/passwd
  -rw-r--r--    1 root     root

# cat /etc/passwd
  User1:x:501:501:User1 full name:/home/test:/bin/bash
```

- User name
- password in `/etc/shadow`
- UID
- Primary GID
- Full user name
- Home directory
- Command shell

`adduser`, `useradd`, `userdel`, `usermod`

User and Groups (3/4)

```
# ls -al /etc/group
  -rw-r--r--    1 root     root

# cat /etc/group
  Grp1:x:500:500, 501      //User1=500, User2=501
```

- Group name
- Password (not used)
- GID
- User list

Commands:

```
cat /etc/group
groupadd, groupdel
groups, groups <groupname>
ls -g
id -g <username>
```

User and Groups (4/4)

```c
int main()
{   // gcc -Wall -o file file.c
    printf ("Real userID =%d, Real grID =%d\n", getuid(), getgid());
    printf ("Effective userID=%d, Effective grID=%d\n", geteuid(), getegid());
    printf ("\nChange group id to 500\n");
    setgid (500); // grp test
    printf ("Real userID =%d, Real grID =%d\n", getuid(), getgid());
    printf ("Effective userID=%d, Effective grID=%d\n", geteuid(), getegid());
    printf ("\nChange user id to 500\n");
    setuid (500); // user test
    printf ("Real userID=%d, Real grID=%d\n", getuid(), getgid());
    printf ("Effective userID=%d, Effective grID=%d\n", geteuid(), getegid());
    return 0;
}
```

```
# ./file
Real userID     =0, Real grID     =0             // 0= root
Effective userID=0, Effective grID=0

Change group id to 500
Real userID     =0, Real grID     =500
Effective userID=0, Effective grID=500

Change user id to 500
Real userID     =500, Real grID     =500
Effective userID=500, Effective grID=500
```

Password

Password file

`/etc/passwd` used to contain the encrypted passwords (shadow passwords)

```
# cat /etc/shadow
  User1:$6$OTGGgx8yzs62RXKbSI8XfEM....dQGMo2Fu7b8MzQfRIFSeKF0Oa.GWbWLE0:15342:0:99999:7:::
```

1. Username - the login name
2. Password: It is the encrypted password. (No `$xx$`=DES, `$1$` = MD5, `$2a$`=Blowfish, `$5$`=SHA256, `$6$` = SHA512, `$y$`=yesencrypt, see `/etc/login.defs` or `/etc/pam.d/common_password: password`
3. Last password change (lastchanged): Days since Jan 1, 1970 that password was last changed

4.-5. The number of days between two password changes

Password

Commands:

```
passwd

On a host, the encryption method (DES, MD5, SHA512, yesencrypt) used is in the file /etc/
login.defs


On NanoPi with busybox, the encryption method used is given by the command:
passwd -help
passwd -a des, or passwd -a md5 or passwd -a sha256 or passwd -a sha512  username
```

File permsissions

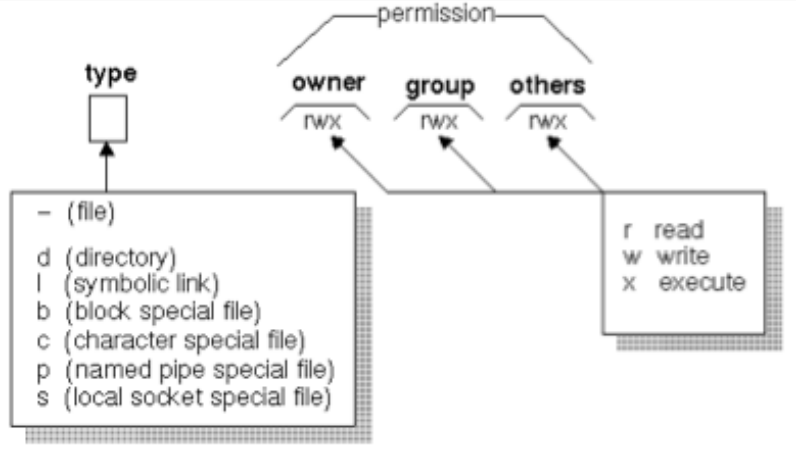
File permsissions

Every file or folder has access permissions. There are three types of permissions (the file or directory rights):

- read access (r)
- write access (w)
- execute access (x)

Permissions are defined for three types of users:

- the owner of the file
- the group(s) that the owner belongs to
- other users



Thus, Linux file permissions are nine bits of information (3 types x 3 type of users), each of them may have just one of two values: allowed or denied.

Files and directories permissions

Access permissions for files and directories mean different things. The table below shows the difference.

Access type	File	Directory
Read	Read the file content	Allow to read the folder content
Write	Modify (create-delete-rename) the file	Allow to create-delete-rename files in the folder
Execute	Execute the file	Allow to go in the folder

```
# chmod 744 file            // this command changes the file permission
# chown test:test file      // this command changes the file owner
```

Files and directories permissions

Remark: if a user is allowed to create-delete-rename files in a directory, he can delete a file in the directory even if this file is write protected (he can't modify this file)

```
whoami
   maeder
ls -al
   drwxrwxr-x.  2 maeder maeder 4096 Dec  9 13:57 testDir

cd testDir
ls -al
   -r--r--r--. 1 maeder maeder    10 Dec  9 14:00 testFile //write protect

rm testFile
   rm: remove write-protected regular empty file 'testFile'? y

ls -al
   TestFile is deleted
```

Set user ID, set group ID, sticky bit (1/2)

In addition to the basic permissions discussed above, there are also three bits of information defined for files in Linux:

- **SUID or setuid: change user ID on execution**. If setuid bit is set, when the file will be executed by a user, the process will have the same rights as the owner of the file, e.g. : program `passwd` which allows a user to change his password must write to the files `/etc/passwd` or `/etc/shadow`. Only root can modify these files. The program passwd is setuid root:

```
-rwsr-xr-x 1 root root 25700 2008-04-08 15:48 /usr/bin/passwd
```

- **SGID or setgid: change group ID on execution**. Same as above, but inherits rights of the group of the owner of the file on execution. For directories it also may mean that when a new file is created in the directory it will inherit the group of the directory (and not of the user who created the file).

Set user ID, set group ID, sticky bit (2/2)

- **Sticky bit** has 2 functions:
  - It was used to trigger process to stay in memory after it is finished (faster restart).
  - A directory that has the sticky bit set allows you to delete a file only if you are the file owner. A few directories on the typical GNU/Linux system have the sticky bit set. For example, the `/tmp` directory, in which any user can place temporary files, has the sticky bit set. This directory is specifically designed to be used by all users, so the directory must be writable by everyone. But it would be bad if one user could delete another user's files, so the sticky bit is set on the directory. Then only the owning user (or `root`, of course) can remove a file

```
$ ls -la test
  -rwxr--r-- 1 root root 0 Nov  2 09:22 test
$ chmod 7744 test                                  // SUID + SGID + sticky bits = 1
$ ls -la test
  -rwsr-Sr-T 1 root root 0 Nov  2 09:22 test       // s or t lower case : with x bit
$ chmod 7644 test
$ ls -la test
  -rwSr-Sr-T 1 root root 0 Nov  2 09:22 test.      // S ot T upper case: without x bit
$ chmod 4644 test
$ ls -la test
  -rwSr-Sr-T 1 root root 0 Nov  2 09:22 test.      // S ot T upper case: without x bit
```

Real and effective user ID and group ID

- Every process has two user IDs: the **effective** user ID and the **real** user ID. (Of course, there's also an effective group ID and real group ID.
- For the test file permissions, Linux uses only the **effective** user ID.

Example:

The `setuid_test` file has the flag SUID set, when the user test executes this file, he has the same right of the owner of the file (root)

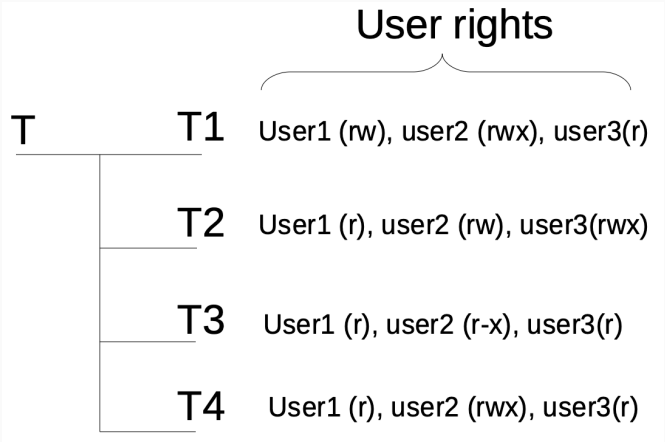
```
$ pwd
  /home/test/work
$ ls -al
  -rwsr-x--x  1 root root 5072 2012-01-11 14:04 setuid_test
$ whoami
test
$ ./setuid_test
  uid=501 euid=0
The real User ID  is test (501) and the effective User ID is
root (0)
```

setuid_test.c:

```c
int main ()
{
  printf ("uid=%d euid=%d\n",
    (int) getuid (),
    (int) geteuid ());
  return 0;
}
// can be tested directly on
the NanoPi
```

ACL (Access Control Lists)

- A directory T has 4 sub directories (T1, T2, T3, T4)
- Each sub-directory has different user rights
- It is not easy (or impossible) to implement this case with standard rights



- The ACL allow a fine tune for the directory-file rights

ACL and the kernel

- Kernel must be compiled in order to allow the ACL

```
$ cd /buildroot
$ make linux-menuconfig
    File systems → The Extended4 (ext4) filesystem → Ext4 PosixAccess Control Lists
```

- The followed filesystems allow the ACL: ext2, ext3, ext4, ReiserFS, JFS, XFS, Btrfs, Tmpfs, JFFS2, CIFS
- In order to allow the ACL, the `/etc/fstab` file must have the acl option:

```
$ cat /etc/fstab
  LABEL=/home             /home                   ext4     rw,acl
```

- If `/etc/fstab` has not the acl option, edit the file `/etc/fstab` and add the directive "acl".

```
$ umount /home
$ mount /home
  or
$ mount -v -o remount /home            //umount and remount /home
```

Add an ACL (1/2)

```
# Add an ACL
$ setfacl -m permissions fileOrDirectory  # Permissions = prefix:[user Or group]:rights
```

Prefix:

- u: : User
- g: : Group
- o: : Other

Rights allowed: r or w or x

Example:

```
$ setfacl -m u::rwx,g::r--,o:--- test                  # is equal to: chmod 740 test
$ setfacl -m u:user1:rw- test                          # user1 can read and write test file
$ setfacl -Rm u:user1:rw TestDirectory                 # -R: Recursive
$ setfacl -m u:user1:rw,u:user2:rwx,g:group1:r,o:--- test   # it is possible to give
                                                       # several rights with the
                                                       # same command
```

Add an ACL (2/2)

Default rights and inheritance:

- A directory has an ACL. The files created in this directory don't inherit its ACL. The inheritance is possible with the -d (default) prefix

```
$ setfacl -m d:u:user1:rw TestDirectory                 # only -u has a default right
$ setfacl -m d:u:user1:rw,d:o:--- TestDirectory         # -u and -o have default rights
  or
$ setfacl -dm u:user1:rw,o:--- TestDirectory            # option -d: all permissions are default rights
```

Delete an ACL:

```
$ setfacl -b test                           # The file test has no ACL
$ setfacl -x u:user1,g:group1 test          # The file test has no rights for user1 and group1
```

Inspect an ACL

```
$ getfacl TestDirectory/
  # file: TestDirectory/
  # owner: op414
  # group: op414
  user::rwx
  user:user1:rwx
  user:user2:r--
  group::rwx
  mask::rwx
  other::---
  default:user::rwx
  default:user:user1:rwx
  default:user:user2:r--
  default:group::rwx
  default:mask::rwx
  default:other::---

$ ls -al TestDirectory/
  rwxrwx---+  TestDirectory            # add a "+" with ACL
```

File attributes

File attributes on ext2, ext3, ext4 filesystems (1/2)

- Extended Filesystem (ext2, ext3, ext4) contains file attributes that are less known, but still very practical
- To view or set these attributes, we have these two commands (in `e2fsprogs`), `lsattr` and `chattr`

```
$ chattr +i file            # add the "i" attribute
$ chattr -i file            # del the "i" attribute
$ chattr =i file            # equal the "i" attribute
$ lsattr file
```

File attributes on ext2, ext3, ext4 filesystems (2/2)

File attributes examples:

Attribute	**Description**
-i	`file/directory` can not be modified, deleted, renamed or linked symbolically, not even by root. Only root or a binary with the necessary rights can set this attribute.
-A	Date of last access is not updated (only useful for reducing disk access)
-S	File is synchronous, the records in the file are made immediately on the disc. (equivalent to the sync option of mount)
-a	File can only be open in append mode for writing (log files, etc.) Only redirection >> can be used, the file can not be deleted.
-c	File is automatically compressed before writing to disk, and unpacked before playback
-d	File will not be saved by the dump command
-j	(ext3 and ext4 only) A file with the `j` attribute has all of its data written to the ext3 or ext4 journal before being written to the file itself
-s	When the file is destroyed, all data blocks are being released to zero.

Finding permissions problems

```
$ find . -perm 200                # file permissions = 200
$ find . -perm -220               # write bit for user and group = 1
$ find . -perm /220               # write bit for user or group = 1
$ find . -perm +220               # write bit for user or group = 1 (like /220)

# "world writable" bit = 1
$ find / -type d -perm -2 -ls     # find the directories with the others can write
$ chmod -R o-w /dir1/dir2         # Be careful: remove other bit = write for all files under /dir1/dir2

# SUID bit = 1
$ find / -type f -perm -4000 -ls
$ chmod u-s /usr/bin/file
$ chmod -R u-s /var/directory/    # Be careful

# GUID bit = 1
$ find / -type f -perm -2000 -ls
$ chmod g-s /usr/bin/file
$ chmod -R g-s /var/directory/    # Be careful

# Sticky bit = 1
$ find / -type f -perm -1000 -ls
$ chmod -t /usr/bin/file
$ chmod -R -t /var/directory/      # Be careful
```

root $PATH with "." - a security risk

For root, it is **one of the biggest security hole** on Linux ("." is the current directory)

```
$ echo $PATH
  .:/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin:
```

Directory `/tmp` has rights 1777, it means that others can write in this directory.

Example: test user writes this script, called `ls`:

```
$ cat ls
#/bin/bash
cp /etc/shadow /tmp/fileasd234
chmod 777 /tmp/fileasd234
exec /bin/ls $argv | grep -v ls
```

what happens if root goes in /tmp and types `ls`?

root $PATH with "." - a security risk

User `test` copies the script to `/tmp`

```
$ ls -al /tmp
  -rwxr-xr-x  1 test test    75 2012-03-09 09:15 ls
```

Problem:

`root` user goes to `/tmp` and executes `ls` command. The problem arises because the `PATH` has the `.` at first position. The wrong `ls` is executed and the shadow file is copied to `/tmp/fileasd234`. Test user can read it.

In order to avoid this flaw, the root path **must not have the .**

The `/tmp` directory can be secured (see next slides)

Secure temporary files (1/4)

- `/tmp`, `/var/tmp` are directories or partitions which hold temporary files
- A typical Linux installation will set `/tmp` rights to **1777**, meaning it has the sticky bit set and is readable, writable, and executable **by all users**.

```
#ls -al /
drwxr-xr-x    2 root     root          4904 Oct 14  2021 bin
drwxr-xr-x    4 root     root          2460 Aug  5 09:56 dev
drwxr-xr-x    8 root     root          3272 Aug  5 09:56 etc
drwxr-xr-x    4 root     root           288 May  7 14:32 home
drwxr-xr-x    5 root     root          2024 Oct 14  2021 lib
lrwxrwxrwx    1 root     root            11 Oct 14  2021 linuxrc -> bin/busybox
drwxr-xr-x    5 root     root           352 Jul  9  2021 mnt
drwxr-xr-x    2 root     root           160 Oct 14  2021 opt
dr-xr-xr-x   46 root     root             0 Jan  1  1970 proc
drwxr-x---    3 root     root           720 Jul 19 16:38 root
drwxr-xr-x    2 root     root          4200 Oct 14  2021 sbin
drwxr-xr-x   12 root     root             0 Jan  1  1970 sys
drwxrwxrwt    3 root     root           220 Aug  5 11:00 tmp
drwxr-xr-x    6 root     root           416 Oct 17  2021 usr
drwxr-xr-x    5 root     root           736 Oct 17  2021 var
```

Secure temporary files (2/4)

- These directories can store temporary bots, malware, rootkits, ...
- A more secure solution would be to set /tmp in its own partition, so that it can be mounted independently of the / partition and have more restrictive options set. An example `/etc/fstab` entry for a /tmp partition might look like:

```
/dev/sda7 /tmp ext4 nosuid,noexec,nodev,rw 0 0
```

–> This would set the `nosuid`, `noexec`, and `nodev` options, meaning:

- no `suid` programs are permitted,
- nothing can be executed,
- and no device files (node file) may exist (see `man mount`).

The `/var/tmp` directory can be deleted and create a symlink pointing to `/tmp`

Secure temporary files (3/4)

The `tmpfs` is a file system which keeps all files in virtual memory. The `tmpfs` uses the `/dev/shm` node file (NanoPi uses `tmpfs`).

In order to improve the security for the `tmpfs`, the `/etc/fstab` file must be modified

```
tmpfs /dev/shm    tmpfs    mode=0777                          0 0
tmpfs /tmp        tmpfs    defaults,nosuid,noexec,nodev,rw    0 0
```

Secure temporary files (4/4)

- Finally, if you don't have the ability to create a fresh /tmp partition on existing drives, you can use the loopback capabilities of the Linux kernel by creating a loopback filesystem that will be mounted as /tmp and can use the same restrictive mount options. (*loopback is a pseudo-device that makes a file accessible as a block device*)
- Create a 1GB loopback filesystem :

```
$ dd if=/dev/zero of=/.tmpfs bs=1024 count=1000000
$ mkfs.ext4 /.tmpfs
$ mkdir /tmp
$ mount -o loop,noexec,nosuid,nodev,rw /.tmpfs /tmp
$ chmod 1777 /tmp
```

After these commands, modify the `/etc/fstab` in order that the loopback filesystem is mounted automatically during the computer boot:

```
/.tmpfs /tmp ext4 loop,nosuid,noexec,nodev,rw 0 0
```

end of presentation