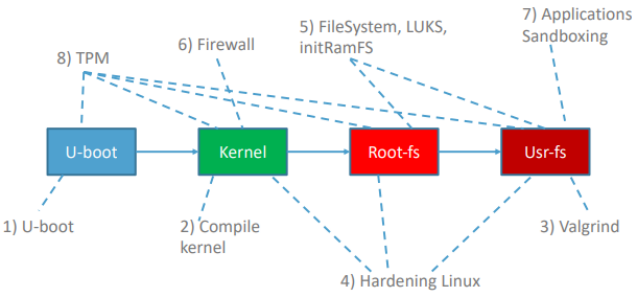


1 Introduction



1.1 Attaques

- 1. Attaques de surface
 - (a) Utilisateurs des ports de debug
 - (b) Connecteurs
 - (c) Alimentations
- 2. Vecteurs d'attaque
 - (a) Réseau (Ethernet, Wifi)
 - (b) Application
 - (c) Port série
 - (d) USB, I2C, Flash, Bluetooth, GPS, etc...

1.2 Compilation pour nanopi

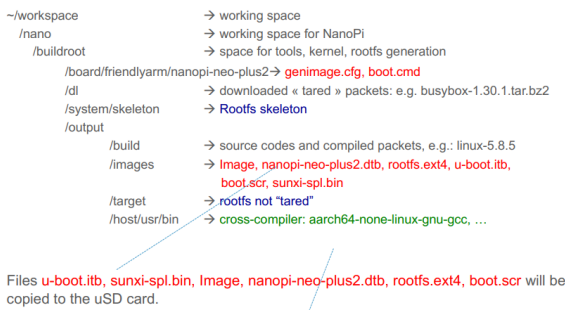
Cross-compilation (ARM) effectuée sur un système x86/x64. Buildroot est le toolchain utilisé. Les éléments suivants sont compilés :

- 1. Bootloader
- 2. Kernel
- 3. Rootfs

Puis les images sont copiées sur la carte SD

2 Buildroot

2.1 Répertoires



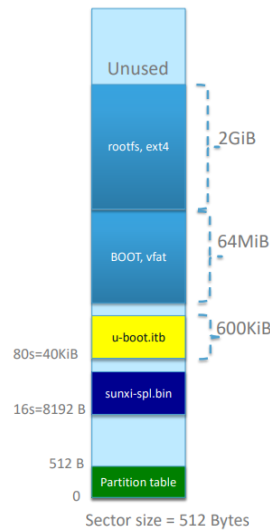
Ce qui est manquant dans le dossier output sera recompilé lorsque la commande `make` est lancée (ou alors en faisant la commande `make <package>-rebuild`. Le dossier `rootfs_overlay` permet d'ajouter des fichiers au rootfs (`/workspace/nano/buildrootboard/friendlyarm/nanopi-neo-plus2/rootfs_overlay`)

2.2 Compilation

Dans le répertoire `buildroot`, effectuer la commande `make menuconfig` puis `make`. `make clean` pour effacer tous les fichiers compilés. La configuration permet notamment de

- 1. Modifier le rootfs

2.3 Carte SD



Area Name	From (Sector #)	To (Sector #)	Size
rootfs, ext4			2GiB
BOOT, vfat			64MiB
U-boot-itb	80		600KiB
sunxi-spl.bin	16	79	32KiB
MBR (partition table)	0	15	512B

`genimage.cfg` → `genimage` → `sdcard.img` → `dd` → carte SD

Les fichiers pour l'initialisation sont

<code>rootfs.ext</code>	Root file system
<code>Image</code>	Noyau Linux
<code>nanopo-neo-plus2.dtb</code>	Flattened device tree
<code>boot.scr</code>	Commandes boot compilées utilis
<code>boot.vfat</code>	Partition boot
<code>u-boot.itb</code>	Boot loader
<code>sunxi-spl.bin</code>	Secondary Program Loader

`boot.vfat` contient `Image`, `nanopi-neo-plus2.dtb` et `boot.scr`. `boot.vfat` (ou `boot.ext4`) permet de créer `BOOT` sur la carte SD

2.3.1 rootfs

Contient `/bin`, `/sbin`, `/root`, `/etc`, etc...

2.3.2 boot.scr

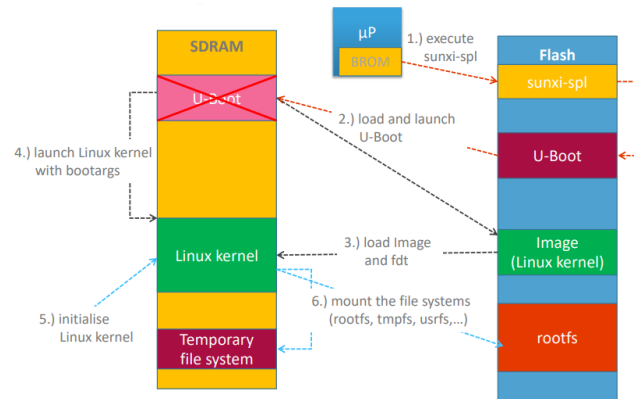
Le fichier `boot.scr` est utilisé par u-boot pour charger le kernel Linux. Il est créé avec la commande `mkimage`

2.3.3 boot.cmd

`boot.cmd` contient des informations de démarrage, notamment les emplacements des différents l'emplacement de `nanopi-neo-plus2.dtb`, du kernel et (si présent) de `l'initramfs`

2.4 Séquence de démarrage (6 phases)

1. Lorsque le μP est mis sous tension, le code stocké dans son BROM va charger dans ses 32KiB de SRAM interne le firmware `sunxi-spl` stocké dans le secteur no 16 de la carte SD / eMMC et l'exécuter.
2. Le firmware `sunxi-spl` (Secondary Program Loader) initialise les couches basses du μP , puis charge l'U-Boot dans la RAM du μP avant de le lancer.
3. L'U-Boot va effectuer les initialisations hardware nécessaires (horloges, contrôleurs, ...) avant de charger l'image non compressées du noyau Linux dans la RAM, le fichier `Image`, ainsi que le fichier de configuration FDT (flattened device tree).
4. L'U-Boot lancera le noyau Linux en lui passant les arguments de boot (`bootargs`)
5. Le noyau Linux procédera à son initialisation sur la base des `bootargs` et des éléments de configuration contenus dans le fichier FDT (`sun50i-h5-nanopi-neo-plus2.dtb`).
6. Le noyau Linux attachera les systèmes de fichiers (`rootfs`, `tmpfs`, `usrfs`, ...) et poursuivra son exécution.



3 U-boot

3.1 Compilation

On configure avec `make ubiit-menuconfig` puis on effectue la compilation avec une des deux manières :

1. `make uboot-rebuild`
2. supprimer les fichiers puis `make`

3.2 Démarrage

Si on appuie sur une touche, on entre en mode u-boot. La commande `booti` permet de lancer l'image linux (`boot` tout court va aussi lancer l'image Linux). Avec les commandes présentes dans `boot.cmd`, on indique l'emplacement dans la ram de `Image` et `nanopi-neo-plus2.dtb`

4 Kernel

4.1 Compilation

On configure avec `make linux-menuconfig` (ou `make linux-xconfig`) puis on lance une compilation avec `make linux-rebuild`

4.2 Busybox

Busybox est un exécutable qui combine beaucoup de fonctions de base (`ls`, `mv`, `rm`, `cat`, etc...). En mettant toutes ces commandes dans un seul programme, on

réduit énormément les redondances et par conséquent la taille de l'exécutable.

On peut également configurer busybox avec `make busybox-menuconfig` puis le compiler avec `make busybox-rebuild`

5 Valgrind

6 Hardening

7 File system

7.1 Génération

Squelette de `rootfs` dans `workspace/nano/buildroot/system`. Il est ensuite copié dans `buildroot/output/target` et les fichiers nécessaires y sont ensuite ajoutés. Une fois que tous les fichiers sont ajoutés, une image `rootfs.xxx` est créé (xxx est ext4, squashfs, etc...)

7.2

7.3 1. De connaître les différents types de systèmes de fichiers ainsi que leurs applications

Pour les systèmes embarqués, il existe deux catégories de systèmes de fichiers : - Volatiles en RAM - Persistants sur des Flash (NOR et de plus en plus NAND)

Deux technologies principales sont disponibles sur les Flash : - soit les MTD (Memory Technology Device) - les MMC/SD-Card (Multi-Media-Card / Secure Digital Card)

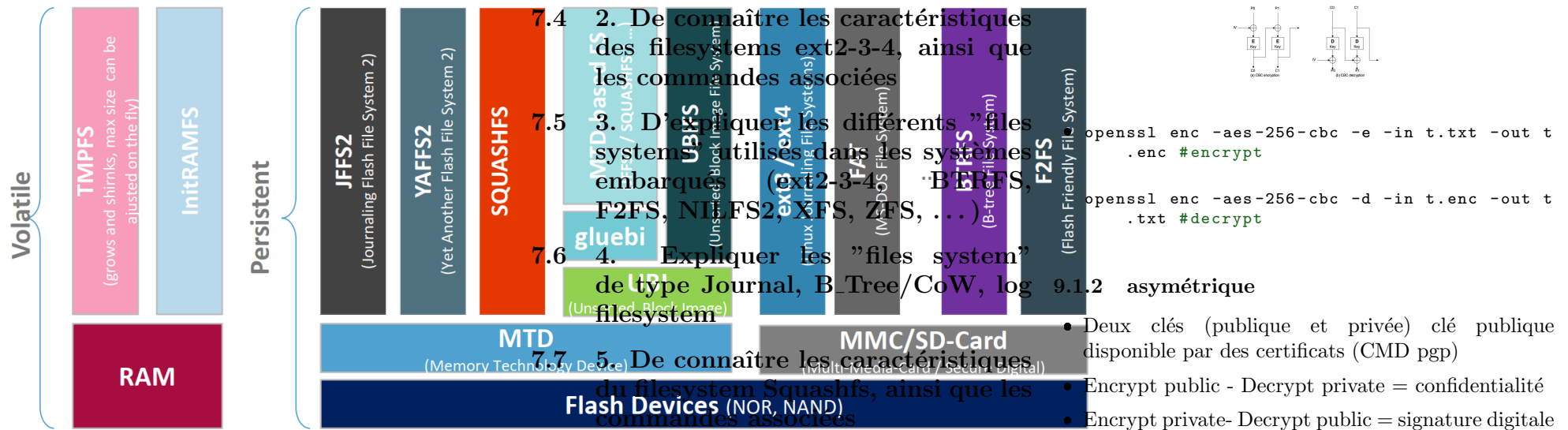


Figure 1: FS type

7.3.1 Choix d'un FS

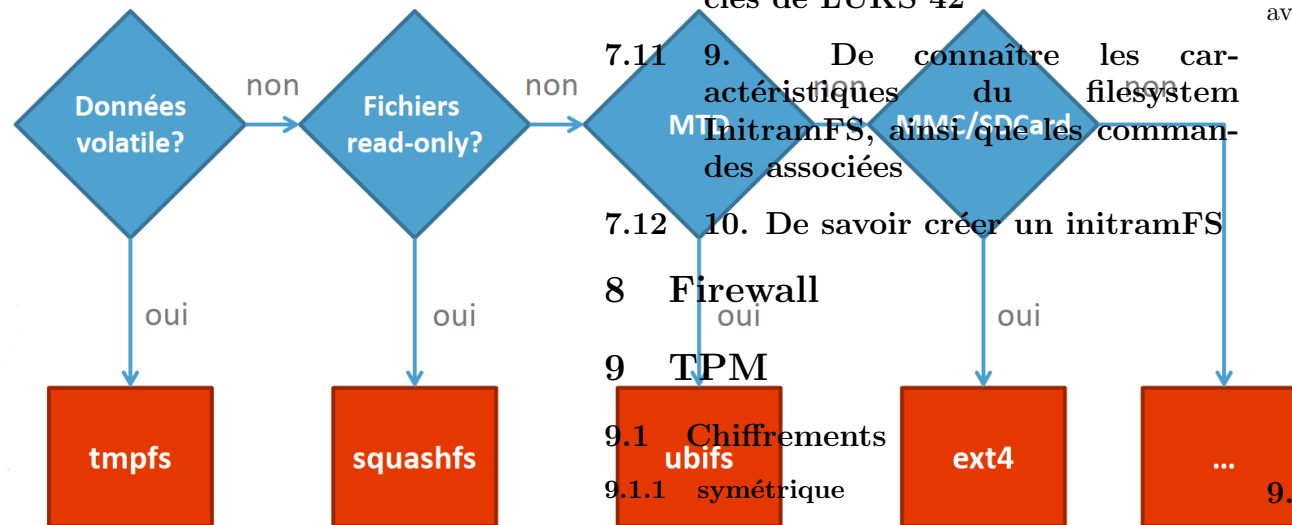


Figure 2: FS type

7.8 6. De connaître les caractéristiques du filesystem tmpfs, ainsi que les commandes associées

7.9 7. De connaître les caractéristiques du filesystem LUKS, ainsi que les commandes associées

7.10 8. Savoir expliquer la gestion des clés de LUKS 42

7.11 9. De connaître les caractéristiques du filesystem InitramFS, ainsi que les commandes associées

7.12 10. De savoir créer un initramFS

8 Firewall

9 TPM

9.1 Chiffrements

9.1.1 symétrique

- Une seule clé pour crypter et décrypter
- Codage par bloc ou par bloc chaîné

9.1.2 asymétrique

- Deux clés (publique et privée) clé publique disponible par des certificats (CMD pgp)
- Encrypt public - Decrypt private = confidentialité
- Encrypt private- Decrypt public = signature digitale

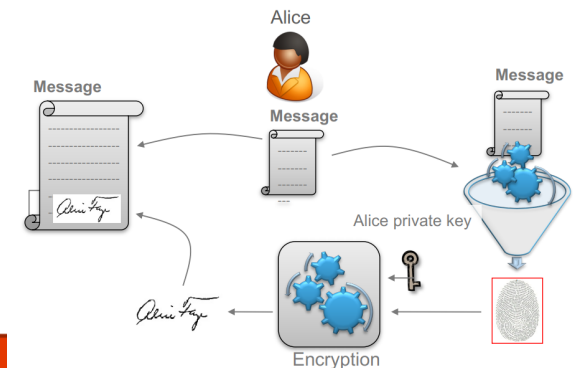
9.1.3 hash

Transforme un texte, document en un nombre de N bits unique (SHA-2, SHA-3, Blake2).

md5sum file => a6a0e8d0522... où
openssl dgst -md5 file

9.1.4 signature

En deux parties: 1. Calcul du HASH puis encryption avec clé privée.

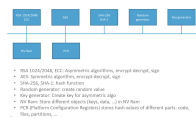


9.2 Implémentations TPM

- discrete : Circuit dédié
- integrated : Partie du μC qui gère le TPM

- Hypervisor : virtuel fournis par personne fiable
- Software : virtuel pour faire des test pas sécurisé

9.3 Architecture interne



9.4 Hiérarchies

- endorsement : réservé au fabricant du TPM et fixé lors de la fabrication.
- platform : réservé au fabricant de l'hôte et peut être modifier par l'équipementier.
- owner : hiérarchie dédiée à l'utilisateur primaire du TPM peut être modifié en tout temps.
- null : réservé aux clés temporaires (RAM s'efface à chaque redémarrage)

9.5 Créer, utiliser clés



9.6 Commandes principales

10 Autres

10.1 Commandes

Commande	Description
netcat (nc)	Couteau suisse du TCP/IP. Permet de scanner des ports
nmap	Analyse des ports ouverts
ssh	Connexion à un système par interpréteur de commande
dd	copie byte à byte entre des streams. (sudo dd if=/dev/zero/ of=/dev/null bs=512 count=100 seek=16)
parted	création / modification de partiatiions (sudo parted /dev/sdb mklabel msdos
mkfs.ext4	commandes ext4 pour créer / modifier une partition

```
tpm2_createprimary -C o -G rsa2048 -c o_prim #
    créer un clé primaire owner
tpm2_getcap handles-transient #voir clé dans
    la RAM
tpm2_getcap handles-persistent #voir clé dans
    la NV-RAM
tpm2_evictcontrol -c o_primary.ctx # sauver
    une clé en NV-RAM
tpm2_flushcontext! -t ##effacer toute la RAM
tpm2_create -C o_prim -G rsa2048 -u child_pub
    -r child_priv #créer clé enfant
tpm2_load -C o_prim -u child_pub -r child_priv
    -c child #charger clé enfant
shred passwd, rm -f passwd #supprimer de l'hôte
```

9.7 encrypter-décrypter, signer-vérifier

```
tpm2_rsaencrypt -c child -s rsaes clearfile -o
    encryptedfile
tpm2_rsadecrypt -c child -s rsaes
    encryptedfile -o clearfile
tpm2_sign -c child -g sha256 -o file.sign file
tpm2_verifysignature -c child -g sha256 -s
    file.sign -m file
```

9.8 Registres PCR

```
tpm2_pcrreset 0
tpm2_pcrextend 0:sha1=8c83...(hash)
```

9.9 Sauver des données sur le TPM

```
tpm2_evictcontrol -c passwd.ctx 0x81010000 -C
    o #sauver
tpm2_unseal -c 0x81010000 > passwd #récupérer
```

9.10 Sauver des données et protéger avec PCR policy

```
sha1sum passwd #calcul hash
tpm2_pcrreset 0 #flush PCR0
tpm2_pcrextend 0:sha1=8c839... #sauve hash
tpm2_createprimary -C o -G rsa2048 -c primary
tpm2_startauthsession -S session
tpm2_policypcr -S session -l sha1:0 -L
    pcr0_policy #créer politique
tpm2_flushcontext session
```

```
tpm2_create -C primary -g sha256 \
    -u passwd_pcr0.pub -r passwd_pcr0.priv \
    -i passwd -L pcr0_policy
```

```
tpm2_evictcontrol -c passwd_pcr0 0x81010000 -C
    o
tpm2_flushcontext session
```

```
shred passwd
rm -f passwd
```

```
tpm2_startauthsession --policy-session -S
    session
tpm2_policypcr -S session -l sha1:0
tpm2_unseal -p session:session -c 0x81010000 >
    passwd
```

10.2 Définitions

Nom	Description
Honeypot	”Pot de miel” ou leurre pour faire croire qu’un système non-sécurisé est présent (à tord)
Toolchain	Codes sources et outils nécessaires pour générer une image exécutable (sur un système embarqué)
Kernel	Coeur Linux (avec le format u-boot)
Rootfs	Root Filesystem (avec tous les dossiers et outils utilisés par Linux)
Usrfs	User Filesystem (applications spécifiques à l’utilisation du système embarqué)
Buildroot	Ensemble de makefiles et patchs qui simplifient et automatisent la création d’un Linux pour système embarqué
uClibc	Librairie c de base similaire à glibc mais plus compacte (pour systèmes MMU-less)
Busybox	Binaire unique qui contient toutes les commandes de base (ls, cat, mv)