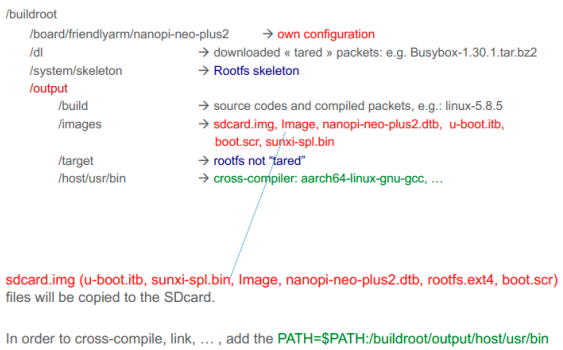
1 Buildroot

1.1 Principaux répértoires



Ce qui est manquant dans le répértoire dans le dossier output sera recompilé lors de la commande `make` ou en précisant le paquet : `make <package>-rebuild` à refaire.

1.2 Principe de fonctionnement

Basé sur des fichiers `make kconfig`, buildroot est un outil qui automatise le process de construction d'un système Linux embarqué en utilisant la cross-compilation. Lors de la commande `make` il va s'occuper de compiler l'entier du système et préparer une image complète et prête pour l'utilisation.

1.3 Configuration pour un hardware donné

Il y a un répertoire situé dans `board/friendlyarm/nanopi-neo-plus2` qui contient plusieurs fichiers intéressants. On y retrouve notamment le fichier `nanopi-neo-plus2/nanopi-neo-plus2.dts` qui contient le FTD ("Flattened Device Tree") qui décrit le hardware. On indique donc à buildroot l'emplacement de ce fichier avec la commande `menuconfig`. Autres fichiers intéressants :

```
root →/buildroot/board/friendlyarm/nanopi-neo-plus2 (main ✗) $ ls
boot.cmd              genimage.cfg         nanopi-neo-plus2.dts  rootfs_overlay
busybox-extras.config linux-extras.config  post-build.sh         ses_linux_defconfig
extlinux.conf         my_patches           readme.txt            uboot-extras.config
```

1.4 Patch buildroot

Il faut spécifier le dossier des patchs `make menuconfig` nous on a fait dans `/nanopi-neo-plus2/my_patches` et un dossier par package à patcher (il faut aussi éditer `ses\-config` pour garder le chemin des patches). Ensuite la technique consiste à profiter de l'outil git et se mettre dans `/buildroot/output/build/uboot-2020.07` et de faire:

```
git init --initial-branch=main
git add .
cd common/
git add .
git commit -m "1st commit"
```

Faire ensuite toutes les modifications à faire, les stager et faire la commande qui créé le patch au format voulu par buildroot:

```
git diff --cached --stat -p > /buildroot/boa⌋
↪ rd/friendlyarm/nanopi-neo-plus2/my_patch⌋
↪ es/uboot/0001_stack_protector.patch
```

On peut ensuite supprimer le paquet en question dans build et il se téléchargera et patchera automatiquement au prochain `make`.

1.5 Configuration de buildroot, u-boot, kernel

Pour buildroot, on utilise `make menuconfig` ça se sauvegarde dans `.config` ou dans `configs/ses_defconfig` (seulement les changement sont dans ce fichier, qui est à la base une copy de `friendlyarm_nanopi_neo_plus2_defconfig` qui se situe au même endroit).
Pour u-boot, on va dans `make menuconfig` catégorie bootloaders et ça va se sauver dans

```
/buildroot/output/build/uboot-xx/configs/nan⌋
↪ opi_neo_plus_defconfig
```

Et dans un fragment files qui ne contient que des petites modifications (save old, make new, use diff, add it to this fragment file):

```
/buildroot/board/friedlyarm/nanopi-neo-plus/⌋
↪ uboot-extras.config
```

Pour le kernel, `make linux-menuconfig`, ça save dans `/buildroot/output/build/linux-xx/defconfig` et nous on le copie et le met dans `ses_linux_defconfig`

1.6 Génération de la carte SD

Pour créer le fichier `sdcard.img` utilise le script `genimage.sh` qui va aller aussi utiliser le fichier `nanopi-neo-plus2/genimage.cfg` qui spécifie les différentes partitions à créer et les images (venant de `output/images`) à mettre dedans.

1.7 Génération du rootfs

Il est copié depuis `/buildroot/system/skeleton` pour le mettre dans `/buildroot/output/target`. Il est ensuite possible d'ajouter des fichiers et des répértoires avec le `/nanopi-neo-plus2/rootfs_overlay`. Après la commande `make` le pseudo rootfs est créé et placé dans un des fichiers images de sorties `rootfs.xxx`

1.8 rootfs_overlay

On peut préprarer la structure des fichiers de la cible dans ce dossier

1.9 Installer un nouveau package dans buildroot

Pour utiliser un package, il faut faire `make menuconfig` et aller choisir le package à aller utiliser. Sinon il faut ajouter le nouveau package "foo" dans le dossier `/buildroot/package`. Il doit contenir au moins:

- Config.in: in kconfig language : on pourra y accéder à travers `make menuconfig`

- foo.mk: fichier qui décrit ou prendre les sources et comment les installer
- optional foo.hash: pour vérifier l'intégrité des fichiers à télécharger
- optional Sxx_foo: le startup script pour le package foo

2 Uboot

2.1 Démarrage du NanoPi

1. Lorsque le uP est mis sous tension, le code stocké dans BROM va charger dans ses 32KiB se SRAM interne le firmware "sunxi-spl" stocké dans le secteur n°16 de la carte SD/eMMC et l'exécuter
2. Le firmware "sunxi-spi" (Secondary Program Loader) initialise les couches basses du uP, puis charge l'U-Boot dans la RAM du uP avant de le lancer
3. L'U-Boot va effectuer les initialisation hardware nécessaires (horloges, contrôleurs, ...) avant de charger l'image non compressée 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 procèdera à son initialisation sur la base des bootargs et des éléments de configuration contenus dans le fichier FDT (sun50i-nanopi-neo-plus2.dtb).
6. Le noyau Linux attachera les systèmes de fichiers (rootfs, tmpfs, usrfs, ...) et poursuivra son exécution

2.2 Principales commandes de Uboot durant le boot

Pour entrer en uboot mode, il faut appuyer sur une touche durant le boot et on arrive dans le prompt u-boot. Ensuite on peut taper help pour avoir une liste des commandes. Les commandes princpales sont

- booti - boot Linux kernel 'Image' format from memory
- ext2load - load binary file from a Ext2 filesystem
- ext2ls - list files in a directory (default /)
- ext4load - load binary file from a Ext4 filesystem
- ext4ls - list files in a directory (default /)
- ext4size - determine a file's size
- fatinfo - print information about filesystem
- fatload - load binary file from a dos filesystem
- fatls - list files in a directory (default /)
- fatmkdir - create a directory
- fatrm - delete a file
- fatsize - determine a file's size
- fatwrite - write file into a dos filesystem
- mmcinfo - display MMC info
- printenv - print environment variables
- setenv - set environment variables

Le fichier `boot.cmd` (tranformé en `boot.scr` par une commande montrée plus bas) contient les commandes que u-boot effecture :

```
setenv bootargs console=ttyS0,115200
↪  earlyprintk root=/dev/mmcblk2p2 rootwait
ext4load mmc 0 $kernel_addr_r Image
ext4load mmc 0 $fdt_addr_r
↪  nanopi-neo-plus2.dtb
booti $kernel_addr_r - $fdt_addr_r
```

La commande `booti $kernel_addr_r - $fdt_addr_r` spécifie l'adresse de l'image, de du fdt et le - précise qu'il n'y a pas de initrd. Voir plus tard avec initramfs où on fera autrement
Création d'un fichier `boot.scr` à partir d'un `boot.cmd`

```
mkimage -C none -A arm64 -T script -d board/
↪  friendlyarm/nanopi-neo-plus2/boot.cmd
↪  output/images/boot.scr
```

2.3 Configuration de uboot

`make uboot-menuconfig` pour configurer (cette config se sauve dans output ubuild uboot-2020.07 .config), `make uboot-rebuild` ou `rm output/build/uboot-xx/.stamp-built` et `make` pour compiler. Après le make 2 fichiers sont créés: `u-boot.itb` et `boot.scr` et se trouvent dans `output/images/`

2.4 Sécurisation de uboot

2 choses sont à faire :

- Retirer les informations de debug, c'est à dire l'option -g lors de la compilation
- Ajouter l'option `-fstatck-protector-strong`. Qui permet d'éviter les attack en buffer overflow, un programme va donc s'arrêter en cas de détection de stack smashing.

Pour mettre l'option de proctection de stack de base dans uboot, il faut faire un patch en éditant le Makefile de uboot de cette manière:

```
KBUILD_CFLAGS += $(call
↪  cc-option,-fstack-protector-strong)
#KBUILD_CFLAGS += $(call
↪  cc-option,-fno-stack-protector)
```

Et ajouter `stackprot.o` qui contient une fonction `__stack_chk_fail` dans le `/common/` et ajouter la ligne `obj-y += stackprot.o` dans le Makefile du common.

2.5 Etapes pour la création de l'image de u-boot.itb

L'image `u-boot.itb` contient 3 parties:

- `u-boot-nodtb.bin` qui est le code de uboot qui a été créé à partir du `elf` de u-boot avec la commande `aarch64-linux-objcopy --gap-fill=0xff` qui fait attention à ne garder que le strict nécessaire (sans debug, symbol, relocation)
- `b131.bin`: trust zone ou ARM Trusted Firmware
- `sun50i-h5-nanopi-neo-plus2.dtb` : Device Tree blob (voir plus bas)

La commande qui fait ça est :
`mkimage -f u-boot.its -E u-boot.itb` (est c'est dans `u-boot.its` (fichier texte) qu'est spécifié les différentes parties de l'image `u-boot.itb`

2.6 Commande strip sur un elf

La commande `aarch64-linux-strip u-boot` (la commande `aarch64-linux-objcopy` le fait automatiquement) supprime les information de symbole et de debug : voici la différence sur le fichier `u-boot.elf`

Diassemble with symbols

```
arm-linux-gnueabihf-objdump -d u-boot
  43e00058 <reset>:
  43e00058:       eb0004f8        bl      43e01440 <save_boot_params_default>
  43e0005c:       e10f0000        mrs     r0, CPSR
  43e00060:       e3c0001f        bic     r0, r0, #31
  43e00064:       e38000d3        orr     r0, r0, #211    ; 0xd3
  43e00068:       e129f000        msr     CPSR_fc, r0
```

Diassemble without symbols

```
arm-linux-gnueabihf-objdump -d u-boot
  43e00058:       eb0004f8        .word   0xeb0004f8
  43e0005c:       e10f0000        .word   0xe10f0000
  43e00060:       e3c0001f        .word   0xe3c0001f
  43e00064:       e38000d3        .word   0xe38000d3
  43e00068:       e129f000        .word   0xe129f000
```

2.7 Création de uImage

uImage est l'image Linux au format uboot elle est créée lors du make. Dans le process du boot.cmd, uboot va spécifier son emplacement dans la variable d'environnement `$kernel_addr_r=0x40080000`

2.8 Flattened Device Tree

Le FTD contient les informations sur le hardware que linux va utiliser pour sa configuration. Le fabricant du uP a surement écrit `sun50i-h5-nanopi-neo-plus2.dts` et on le passe en `.dtb` avec la commande `dtc board.dts {o board.dtb`. Lors du boot.cmd on précise l'adresse de cd `.dtb` dans la variable d'environnement `$fdt_addr_r=0x4FA00000`

2.9 Mapping de la SDCard

La carte SD contient ces éléments

- `sunxi-spl.bin`: Secondary Program Loader
- `u-boot.itb`: selon plus haut
- `boot`: image `.ext4` qui contient 3 autres choses: `Image`, `nanopi-neo-plus2.dtb` et `boot.scr`
- `rootfs`: le filesystem de linux

2.10 boot.scr

u-boot au démarrage, sans pression d'une touche, va chercher le fichier `boot.scr` (format boot script) et l'exécuter. Il contient enfait les instructions du fichier `boot.cmd`.

3 Kernel

3.1 Principaux répertoire du noyau linux

Les principaux répertoires du kernel sont listés:

- arch Hardware dependent code
- block Generic functions for the block devices
- crypto Cryptographic algo. used in the kernel
- Documentation Documentation about the kernel
- drivers All drivers known by the kernel
- fs All filesystem know by the kernel
- include kernel include files
- init Init code (function start_kernel)
- ipc Interprocess communication
- kernel Kernel code, scheduler, mutex, ...
- lib different libraries used by the kernel
- mm Memory management
- net Different protocols, IPv4, IPv6, bluetooth, ...
- samples Different examples, kobject, kfifo, ...
- security Encrypted keys, SELinux, ...
- sound Sound support for Linux kernel
- virt Kernel-based virtual machine

3.2 Principales méthodes pour sécuriser le noyau

Pour configurer linux : `make linux-menuconfig` ou `make linux-xconfig`. La commande `sysctl` permet de voir les paramètres du kernel

- ne pas inclure le Linux kernel `.config` dans le kernel : `Kernel .config support`
- protéger la stack en activant la détection de buffer overflow : General architecture-dependant options : [*] Stack Protector buffer overflow detection, [*] Strong Stack Protector
- randomize_va_space: avoir des adresse aléatoire pour la stack si 1 avec 2 ça ajoute le heap random. Pour le faire :

```
echo 2 >
↪ /proc/sys/kernel/randomize_va_space
sysctl -w kernel.randomize_va_space=2
```

- randomize slab allocator (allocation des kernel obects.o):

```
[*] Allow slab caches to be merged
[*] Randomize slab freelist
[*] Harden slab freelist metadata
[*] Page allocator randomization
```

- randomize address of kernel imge

```
Kernel feature => [*] Randomize the
↪  address of the kernel image
```

- mettre le kernel code read-only

```
Kernel Feature => [*] Apply r/o
↪  permissions of VM areas also to their
↪  linear aliases
```

- optimiser le kernel pour la performance (gcc -O2) plutôt que pour la taille (gcc -Os)

```
General Setup => Compiler optimization
↪  level  [*] Optimize for performance
```

- random number generator: si présent sur le hardware

```
Device drivers => Character Devices =>
↪  [*] Hardware Random Number Generator
↪  Core support
[*] Timer IOMEM HW Random Number
↪  Generator support
[*] Broadcom BCM2835/BCM63xx Random
↪  Number Generator support
```

  sinon utiliser le paquet haveged (dans buildroot menuconfig, miscellaneous). Pour voir l'entropy: `cat /proc/sys/kernel/random/entropy-avail`

- l'emplacement `/dev/mem` contient des infos sur kernel memory et process memory. Il faudrait éviter l'accès à d'autres utilisateurs que le root:

```
Kernel Hacking => [*] Filter access to
↪  /dev/mem
=> [*] Filter I/O access to /dev/mem
```

- strip l'assembleur du kernel durant le link pour rendre le reverse du code encore plus dure

```
Kernel Hacking => Compile time check and
↪  compiler options
=> [ ] Compile the kernel with debug info
=> [*] Strip assembler-generated symbols
↪  during link
```

- diminuer l'accès au kernel syslog

```
Security options => [*] Restrict
↪  unprivileged access to the kernel
↪  syslog
```

- initialiser tout le stack et le heap avec des zéros

```
Security options => Kernel Hardening
↪  Options => Memory Initialization
=> Initialize Kernel Stack =>no automatic
↪  initialization (weakest) (if possible)
[*] Enable heap memory zeroing on
↪  allocation by default
[*] Enable heap memory zeroing on free by
↪  default
```

- empêcher la copie depuis/vers la mémoire du kernel qui est douteuse (taille de mémoire plus grand que l'objet)

```
Security options
=> [*] Harden memory copies between
↪  kernel and userspace
[*] Allow usercopy whitelist violations
↪  to fallback to object size
[*] Harden common str/mem functions
↪  against buffer overflows
```

- un soft en python, régulièrement mis à jour peut tester la config du kernel : `kconfig-hardened-check.py`

3.3 Software attacks

3.3.1 Buffer overflow

L'attaque consiste à utiliser programme qui va aller écrire dans la stack d'un programme, qui est exécutable. On peut par exemple insérér et exécuter un shell code dans une partie exécutable de la stack. Et ensuite on peut faire ce qu'on veut.

3.3.2 ret2libc

On va aller overflow l'adresse de retour d'une fonction avec l'adresse de system() qui est une fonction standard C library (libc). Elle permet d'exécuter une commande en shell. On va donc tenter d'appeler le bash: `/bin/sh`

3.3.3 ROP

Return-oriented programming: on va utiliser plein de petits bouts de codes (gadgets) dans toutes les libraries disponibles et on essaie de faire un programme malicieux. ASLR ou le PIE permettent d'éviter ça.

3.4 Protections

3.4.1 ASLR

ASLR (Adress Space Layout Randomization) randomize_va_space randomizes the stack and heap addresses

3.4.2 PIE

Le PIE (Position Independent Executable) fait en sorte que l'adresse des codes change.

3.4.3 Canary

C'est une valeur qu'on place à la limite d'un buffer. Si elle est écrasée, on peut savoir qu'il y a eu un buffer overflow.

4 Valgrind

Valgrind a 5 outils:

- Memcheck: memory error detector (adresse d'une autre stack, variable non-initialisée)
- Cachegrind: cache and branch-prediction profiler (indexation peu optimisée d'un tableau)
- Callgrind: call-graph generator
- Helgrind: thread error detector
- Massif: Heap and stack profiler (allocation sans free) pour voir: `ms_print massif.out.13866`

5 Hardening Linux

5.1 Contôler l'intégrité d'un package, d'un programme

En premier télécharger le programme foo et sa signature ensuite il faut vérifier l'intégrité avec

```
gpg --verify foo.tar.gz.asc
```

Il nous répond qu'il peut pas en donnant une RSA key: par ex. 23948UT5. Donc il faut importer cette clé publique:

```
gpg --keyserver keyserver.ubuntu.com
↪  --search-keys 23948UT5
```

Et relancer la vérification et c'est bon.

5.2 Configurer un nouveau package, programme

Détarer le package et afficher les options de compilationss:

```
tar xvzf foo.tar.gz
cd foo
./configure --help
```

Si il n'existe pas encore dans buildroot (donc une version inférieure n'a jamais été installé) on le place dans `/buildroot/packages`. Et on va faire les choix de versions dans le `foo.mk` et ensuite on le configure avec `make menuconfig` le choix de config sont décris dans `Config.in`

5.3 Cross-compiler un programme

5.4 Contrôler les services, les ports ouverts

Les services sont lancé au démarrage si leur Startup scripts se trouve dans le `/etc/init.d` Le script `rcS` commence les autres scripts (S01..., S02..., not rcK) dans l'ordre alphabétique. On essaie d'avoir le moins de de services possibles. On est obligé de créer un script pour démarrer automatiquement un nouveau service. Si on supprime un S0.... le service ne sera pas débuté au prochain startup. Pour voir les services en en cours: `ps -ale`, le tree: `ps -axjf`, leurs privilèges: `ps aux`, montrer les fichiers ouverts et leurs privilèges: `lsof | grep sshd`. Pour contrôler les ports ouverts, `netstat -atun` montre les tcp and udp, `lso | grep IP` montre les fichiers ouverts et `nmap` (scan all tcp ports for the host 192.168.0.11) et (// scan all udp ports for the host 192.168.0.11)

```
nmap -Pn -p 1-65535 192.168.0.11
nmap -sU -Pn -p 1-65535 192.168.0.11
```

5.5 Contrôler les files systems

Leur structure se trouve dans `/etc/stab`

5.6 Contrôler les permission des fichiers, répertoires

Les users normaux devraient pas avoir accès au syslog files `/var/log`. Aucun fichier ou repertoires writable devrait exister, à part pour les temporary directories `/tmp`, `/usr/tmp` et `/var/tmp` (`ls /` ou `find /etc -type d -perm -0002 -print`) Les fichiers dans `/etc` must not have group and other bits writable (`find /etc -type f -perm -0002 -print` ou `find /etc -type f -perm -0020 -print`). SUID ou SGID binaries devraient être désactivés: (`find / -perm -4000 -print` ou `find / -perm -2000 -print`). Faudrait forcer `/tmp` à être cleaned durant le shutdown et encore mieux si possible: on met `/tmp` sur un disque en ram

5.7 Sécuriser le réseau

De base, trop d'options sont actvées et donnent trop d'information au hacker.
Disable IPV6

```
echo 1 >
↪  /proc/sys/net/ipv6/conf/all/disable_ipv6
echo 1 > /proc/sys/net/ipv6/conf/default/dis
↪  able_ipv6
Or sysctl -w
↪  /net/ipv6/conf/default/disable_ipv6=1
```

IP source routing must be disabled

```
echo 0 > cat /proc/sys/net/ipv4/conf/all/acc
↪  ept_source_route
Or sysctl -w
↪  net/ipv4/conf/all/accept_source_route=0
```

Forwarding (Routing) of normal and multicast packets should also be deactivated unless expressively needed

```
echo 0 >
↪  /proc/sys/net/ipv4/conf/all/forwarding
echo 0 >
↪  /proc/sys/net/ipv6/conf/all/forwarding
echo 0 >
↪  /proc/sys/net/ipv4/conf/all/mc_forwarding
echo 0 >
↪  /proc/sys/net/ipv6/conf/all/mc_forwarding
Or sysctl -w net/ipv4/conf/all/forwarding=0
sysctl -w net/ipv6/conf/all/forwarding=0
sysctl -w net/ipv4/conf/all/mc_forwarding=0
sysctl -w net/ipv6/conf/all/mc_forwarding=0
```

Block ICMP redirect messages

```
echo 0 > /proc/sys/net/ipv4/conf/all/accept_
↪  redirects
echo 0 > /proc/sys/net/ipv6/conf/all/accept_
↪  redirects
echo 0 > /proc/sys/net/ipv4/conf/all/secure_
↪  redirects
echo 0 > /proc/sys/net/ipv4/conf/all/send_re
↪  directs
Or sysctl -w
↪  net/ipv4/conf/all/accept_redirects=0
sysctl -w
↪  net/ipv6/conf/all/accept_redirects=0
sysctl -w
↪  net/ipv4/conf/all/secure_redirects=0
sysctl -w net/ipv4/conf/all/send_redirects =0
```

Enable source route verification in order to prevent spoofing

```
echo 1 >
↪ /proc/sys/net/ipv4/conf/default/rp_filter
Or sysctl -w
↪ net/ipv4/conf/default/rp_filter=0
```

Log all malformed packed and ignore icmp bogus ones

```
echo 1 >
↪ /proc/sys/net/ipv4/conf/all/log_martians
echo 1 > /proc/sys/net/ipv4/icmp_ignore_bogu⌋
↪ s_error_responses
Or sysctl -w net/ipv4/conf/all/log_martians=1
sysctl -w net/ipv4/icmp_ignore_bogus_error_r⌋
↪ esponses=1
```

Disable ICMP echo and timestamp responses sent via broadcast or multicast

```
echo 1 > /proc/sys/net/ipv4/icmp_echo_ignore⌋
↪ _broadcasts
Or sysctl -w
↪ net/ipv4/icmp_echo_ignore_broadcasts=1
```

Increase resilience under heavy TCP load by increasing backlog buffer and by enabling syn cookies

```
echo 4096 >
↪ /proc/sys/net/ipv4/tcp_max_syn_backlog
echo 1 > /proc/sys/net/ipv4/tcp_syncookies
Or sysctl -w
↪ net/ipv4/tcp_max_syn_backlog=4096
sysctl -w net/ipv4/tcp_syncookies=1
```

la commande `systctl -p` montre le contenu `/etc/sysctl.conf`

5.8 Contrôler-sécuriser les comptes utilisateurs

Ils sont affichés dans `/etc/shadow`. Le umask(user file creation right) doit être 0027 dans `cat /etc/profile.d/umask.sh` et ça met 0 dans les others. On peut aussi ajouter un message disant que c'est interdit avant le login dans le fichier `/etc/issue`

5.9 Limiter le login root

Le root devrait être uniquement possible par console : `echo "ttyS0" > /etc/securetty`. Seulement le root peut accéder au dir root: `chmod 700/root`. On utiliser `sudo` pour prendre les droits de roots. Le PATH du root ne doit pas contenir le current dir ou des writable dir: `PATH not equal .:/tmp:/var/tmp: etc`

5.10 Sécuriser le noyau

ASLR (comme en dessus) +write protect kernel make linux-config

```
Kernel Feature => [*] Apply r/o permissions
↪ of VM areas also to their linear aliases
General Setup => [*] Set loadable kernel
↪ module data as NX and text as RO
```

Strip-assembler comme plus haut, -fstack-protector, restrict unprivilegiend access to kernel syslog

5.11 Sécuriser une application

Ajouter les `CFLAGS="-fPIE -fstack-protector-all -D\_FORTIFY\_SOURCE=2 -ftrapv"` et `LDFLAGS="-Wl,-z,now,-z,relro -z,noexecstack, -pie "` dans le Makefike.

- PIE: position independant executable pour avoir ROP
- -fstack-protector-all: ajout de canary random
- D_FORTIFY_SOURCE=2: overflow check for memcpy, ...
- ftrapv: integer (only) overflow activate a catcheable signal SIGABRT
- -Wl,-z,now,-z,relro: tous les symbols dynamiques sont résolut durant le début du porgramme et ça empêche des overwrite attacks

5.12 Contrôler le démarrage de Linux

5.13 Méthodologie OSSTMM simplifiée

1. Access: compter le nombre d'accès possible, le nombre de port TCP/UDP ouverts (plus haut) + `netsat -atunp`
2. Authentification: compter le nombre d'accès (ssh, console, ...) avec username et password
3. Confidentiality: compter le nombre de connexion où les données sont cryptées (ssh est mieux telnet)
4. Vulnerability, Weekness, Concern: check si un program a des vulnérabilité sur internet, tester les passwords avec les hashcat, .. et tester la robustesse des algorithme de cryptographie
5. Exposure: Ne pas donner la version de openssh sur nmap, c'est débile

6 FileSystem

6.1 Différents types de systèmes de fichiers et leur applications

Il y a 2 catégories: RAM et Flash. 2 catégories de flash, MTD et MMC/SD-Card. En volatile on a: TMPFS et initRAMFS. En MTD (plus ancien), on a JFFS2, YAFFS2 et (SQUASHFS), UBIFS. En MMC/SD-Card, on a EXT3-4, FAT, BTRFS, NILFS, (SQUASHFS), XFS, ZFS et F2FS. En MMC on 3 couches: l'interface, le Flash Translation Layer (bad block mgmnt, garbage collection et wear levelling) et la zone de stockage. MMC/SD-Card a le partition alignement qui est critique, les données sont écrites par blocks ou pages. Pour assurer l'alignement, le début de secteur de chaque partition doit être un multiple de $2^20 (= 1048576)$ Bytes.

6.2 Ext2-3-4

A part le 2, ils sont tous journalisés et ils sont rétrocompatibles (référence pour embedded system). Les différentes commandes pour gérer les .ext4 :

```
# Create a partitions (rootfs), start 64MB,
↪  length 256MB
sudo parted /dev/sdb mkpart primary ext4
↪  131072s 655359s
# Format the partition with the volume label
↪  = rootfs
sudo mkfs.ext4 /dev/sdb1 -L rootfs
# Modify (on the fly) the ext4 configuration
sudo tune2fs <options> /dev/sdb1
# Check the ext4 configuration
mount
sudo tune2fs -l /dev/sdb1
sudo dumpe2fs /dev/sdb1
# mount an ext4 file system
mount -t  ext4 /dev/sdb1 /mnt/test // with
↪  default options
```

Pour pouvoir utiliser mkfs.ext2 et tune2fs, il faut configuer make busybox-menuconfig, Les principales options sont

- noatime, nodiratime, relatime pour réduire les write non-nécessaires
- Data=journal (journal avant écriture) safest
- Data=ordered (data d'abord) de base
- Data=writeback (pas d'ordre)
- acl : support POSIX Access Control Lists
- default: options qu'on met dans `/etc/fstab`

6.3 Différents FS

6.3.1 BTRFS

B-Tree filesystem

6.3.2 F2FS

Flash-Friendly File System, log filesystem with many parameter et fonctionne bien sur beaucoup de support

6.3.3 NILFS2

New Implementation of a Log-structured File System. Log and B-tree avec un userspace garbage collector

6.3.4 XFS

Journaling filesystem, support huge filesystems mais ne support pas bien la perte en mode standby

6.3.5 ZFS

Zettabyte (10^21) File System. B-tree with strong data integrity, support huge fs mais pas fait pour de l'embarqué (lot of ram)

6.4 Journal, B_tree/Cow, log filesystem

6.4.1 Journal

Garde une trace de toutes les modifications dans un journal. Cela permet de restored corrupted file. Les modifs vont d'abord dans le journal ensuite dans le disk. En cas de corruption: soit le journal est consistent (on replay le journal au mount time), soit il l'est pas et on drop les modifications

6.4.2 B-Tree/CoW

Les datas structure génèrent des binary tree. Cow pour copy on write: on ne modifie pas tout de suite l'original et on fait un copie plus loin et on attend le commit. Si il y a un interruption, on garde l'ancien.

6.4.3 Log filesystem

Use the storage medium as circular buffer et on écrit toujours à la fin (et on garde compacte). C'est souvent utilsé pour les flash medias. C'est un cow spécifique

6.5 SQUASHFS

C'est un compressed, read-only fs use in embedded system

```
#Create the squashed file system dir.sqsh
↪  for the regular directory /data/config/
mksquashfs /data/config/ /data/dir.sqsh
# mount
mount -o loop -t squashfs /data/dir.sqsh
↪  /mnt/dir
#it is possible to copy the dir.sqsh to an
↪  unmount part with dd
umount /dev/sdb2
dd if=dir.sqsh of=/dev/sdb1
# and mount the partition as squashfs
mount /dev/sdb2 /mnt/dir -t squashfs
```

make menuconfig : target pkg, fs and flash : choose squashfs

6.6 TMPFS

Dynamic FS qui keep tous les files dans une virtual memory. Il met tout dans le cache interne du kernel et il grow and shrinks selon. On peut ajuster sa taille limite. Par rapport à ramfs on peut resize et swaper. Il doit être mount at /dev/shm (fstab). On mount aussi /tmp et /var comme tmpfs

6.7 LUKSFS

Les datas dans la partition luks sont cryptée. Les datas qui sont utilisées dans le userspace sont décryptée par dmcrypt. Linux Unified Key Setup est le standard Linux pour encrypter un hard disk. Toutes les infos de setup sont stockées dans le header pour une meilleure portabilité. dmcrypt crypts an entire partition. Avantages de luks: comptability via standardization, secure against attacks, support for multiple keys, effective passphrase revocation and free. cryptsetup est un utilitaire pour configurer dmcrypt qui utilise les nodes files /dev/random et /dev/urandom. dmcrypt (device-mapper) crypts target and provides transparent encryption of block en utilsant le kernel crypto api. L'user peut spécifier une ciphers symmtetric, un mode

d'encryption, une clé, ... et surtout un new block device dans le /dev/mapper. Toute donnes écrites à ce device seront cryptées, et lues seront décryptées. Pour enable dmcrypt, make linux-menuconfig, device driver, multiple devices drivers support, device mapper support crypt target support et mettre ¡*¿. cryptsetup est un package à ajouter dans menuconfig.

```
# creation d'une passphrase et d'un fichier
dd if=/dev/urandom
↪  of=/buildroot/output/images/passphrase
↪  bs=1 count=64
dd if=/dev/zero
↪  of=/buildroot/output/images/part3.luks
↪  bs=1G count=2
#create a LUKS partition
echo YES | sudo cryptsetup --debug --pbkdf
↪  pbkdf2 luksFormat
/buildroot/output/images/part3.luks
↪  /buildroot/output/images/passphrase
#dump the header information
sudo cryptsetup luksDump
↪  /buildroot/output/images/part3.luks
#create a mapping /dev/mapper/usrfs1
cryptsetup open --debug --type luks
↪  --key-file
/buildroot/output/images/passphrase
↪  /buildroot/output/images/part3.luks
↪  usrfs1
#format the LUKS partition as EXT4
mkfs.ext4 -L luks /dev/mapper/usrfs1
#mount the LUKS partition to /mnt/usrfs
mkdir /mnt/usrfs
mount /dev/mapper/usrfs1 /mnt/usrfs
#copy rootfs files to LUKS partition
cp /buildroot/output/images/rootfs.ext4
↪  /mnt/usrfs/
#unmount the LUKS partition and remove
↪  existing mapping
umount /mnt/usrfs
cryptsetup close usrfs1
```

Pour permettre au Nanopi de connaitre la passphrase, ajouter ça au post-build.sh avant le umount:

```
cp /buildroot/output/images/passphrase
↪  /mnt/passphrase
```

Cette passphrase doit être ajouter au fichier de boot.ext4 dans gen-image.cfg et il faut aussi ajouter la partitions luks

```
partition part3 {
partition-type = 0x83
image = "part3.luks"
}
```

Pour faire tout ça au démarrage du Nanopi: /etc/init.d/s40luks

```
# start()
echo -n "Mounting luks rootfs... "
echo "Mounting boot partition"
mount /dev/mmcblk2p1 /mnt
echo -n "Unlocking luks rootfs"
cryptsetup open --type luks --key-file
↪  /mnt/passphrase /dev/mmcblk2p3
usrfs1
echo "Unmounting boot partition"
umount /mnt
echo "Mounting part3"
mount /dev/mapper/usrfs1 /mnt
mount /mnt/rootfs.ext4 /mnt
echo "LUKS rootfs ready"
# stop()
echo -n "Unmounting luks rootfs"
umount /mnt
umount /mnt
cryptsetup close usrfs1
```

6.8 LUKS keys

Il y a une master key qui est générée à l'initialisation. La passphrase passe à travers un algo (pbkdf2) et vient se combiner avec la master key dans un "crypt master key cipher" et tout ça donne une encrypted master key qui va être stored. Pour crypt ou decrypt une partition on a de nouveau besoin de la master key et de la passphrase. On peut ajouter plusieurs passphrase.

6.9 InitramFS

Permet d'effectuer des tâches le plus vite possible à travers un script /init qui peut s'occuper de copier et gérer les partitions luks. Initrams est cpio archive file, il peut être générer à la main et il contient at least one file called /init qui va être lancer par le kernel. Il a des commandes très basique: mount, exec, ...

6.10 Création d'un initramFS

On PC, NanoPi rootfs is in this directory: /buildroot/output/target/
Principle to build an initramfs :

1. Copy the right files into a directory ($HOME/ramfs)
2. Copy these files in a cpio archive file
3. Compress this file
4. Add the uboot header

7 FileSystemSecurity

7.1 Files permissions

Chaque user a un UID et chaque group un GID. Les lists sont `/etc/passwd` et `/etc/group`. Il y a 3 types de permissions: read access (r), write access (2) et execute (access). Et c'est défini pour 3 type de user: owner,group,others (9bits: 3x rwx).

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
```

SUID or setuid: change user id on execution to owner (-rws on owner). SGID or setgid: change group ID on execution. Sticky bit: 2 fonctions, 1) process stay

in memory after it finished et 2) a directory avec sticky bit permet de supprimer un fichier si on est le file owner (ça sert dans `/tmp` on run). Pour mettre un le sticky bit on fait chmod 4755

7.2 Contrôler et sécuriser les comptes

Pour créer un user: `adduser mdp` de base c'est du md5 mais on peut configurer pour que ça soit du sha512. Pour changer un mdp: `passwd -a sha512 test1` et pour vérifier les encryptions: `cat /etc/shadow` (name, 6 pour sha512, 1 pour md5, *saltvalue* et *passwordCrypt*).

7.3 Real and effective user id and group id

Le user id réel c'est celui qu'on est `whoami` et le effective c'est celui qu'on prendrait avec le suid.

7.4 ACL

Access control list qui permet de fine tune les droits dans un directory. Il faut l'ajouter dans make linux-menuconfig. Ext2, ext3, ext4, ReiserFS, JFS, XFS, Btrfs, Tmpfs, JFFS2 et CIFS ont le ACL. On doit ajouter l'option au fstab lors du montage. Commande:

```
setfacl -m permissions fileOrDirectory
# example
setfacl -m u::rwx,g::r--,o:--- test // is
↪  equal to: chmod 740 test
# delete acl
setfacl -b test
# voir acl
getfacl TestDirectory/
```

7.5 Attributs des fs ext2-4

```
#set
chattr -i file
#get
lsattr file
```

Attributs: -i (peut pas être modifié, supprimé, renommé or lié même pas par le root), -A (date of access is not updated), -S (synchronous), -a (only in append mode), -c (auto compress), -d (not saved by the dump), -j (write to journal before data) et -s (quand le file est destroy, toutes les datas vont à 0)

7.6 Rechercher les permissions de fichier faibles

```
#file permissions = 200
find . -perm 200
#rechercher/retirer le setuid
find / -type f -perm -4000 -ls
chmod u-s /usr/bin/file
chmod -R u-s /var/directory/ // Be careful
```