Zabezpečení embedded systému proti degradaci úložiště

Marek Vašut <marex@denx.de>

October 5, 2014

Marek Vasut

- Software engineer pro DENX S.E. od r. 2011
  - Embedded and Real-Time Systems Services, Linux kernel and driver development, U-Boot development, consulting, training.
- Linux kernel hacker
- Custodian @ U-Boot bootloader

Osnova

- Boot process detailně
- Hardwarové problémy
- Softwarová ochrana proti poškozní dat
- Softwarová ochrana proti změně dat

Bootovací process

- Power on nebo Reset
- CPU spouští kód od předem definované adresy
- Bootloader je spuštěn
- Kernel je spuštěn
- Root filesystem se začne používat

Právě mezi těmito kroky se schovávají problémy.

Power on nebo Reset

- Nejdříve se děje HW magie
- Relevantní komponenty jsou vytaženy z resetu
- Komponenty jsou tudíž v definovaném stavu
- Až pak může CPU začít spouštět kód

Reset routing

- Opakující se problém!
- Reset není připojen ke všem komponentům
- Častý případ jsou MTD zařízení (SPI NOR) nebo SD/MMC
- Příklad: i.MX23 bootuje z 64 MiB SPI NOR
  - i.MX23 BootROM použije READ opcode s 3-byte addresováním
  - SPI NOR nepodporuje opcody s 4-byte adresováním
  - Linux přepne SPI NOR do 4-byte adresovacího režimu
  - Linux používá standardní (3b) opcody se 4-byte adresou

  $\rightarrow$ Nastane reset

  $\Rightarrow$ Zařízení nenastaruje – Proč?
- Naivní řešení: Poslat RESET opcode ze software (NELZE!)
- Řešení: CPU má Reset-OUT:
  - Připojit k Reset-In bootovacího media
  - RstOut není k dispozici $\Rightarrow$ externí resetovací obvod
  - RstOut nesplňuje časování $\Rightarrow$ externí resetovací obvod

CPU spouští kód

- První kód který CPU spouští
- Může běžet kód interně z CPU (BootROM)
- Může běžet kód externě z paměti (NOR, FPGA, . . .)

BootROM:

- Často closed source
- Většinou nelze aktualizovat nebo opravit (ROM)
- Zprostředkovává načítání z netriviálních medií (SPI NOR, SD/MMC, RAW NAND, USB, Síť, . . .)

Tipy pro bootvanání

- Resetujte z PWR/HOT/COLD
- SPI NOR:
  - Zajistit, že nHOLD je korektně pullnutý (pokud GPIO)
- SD/eSD/MMC/eMMC:
  - Pozor na chování na konci životnosti
    $\rightarrow$ Musí indikovat chybné bloky, ne vracet špatná data
- NAND:
  - Výrobce garantuje, že první sektor je vždy OK
    - Nemusí se vztahovat na přepis prvního sektoru!
    - Nutno pozorně číst datasheet !
  - První sektor je 1/2/4 KiB $\Rightarrow$ U-Boot SPL
  - Pozor na MLC NAND

U-Boot SPL

U-Boot SPL:

- První uživatelský kód
- Mnohem menší než U-Boot
- Funkcionalita v závislosti na zařízení
- Provede základní inicializaci HW
- Načte, ověří a spustí další stupeň
  $\rightarrow$ Další stupeň je U-Boot, Linux, ...

Specifika RAW NAND:

- Plná podpora pro UBI se do 4KiB nevejde
- U-Boot SPL dělá ECC, ale neaktualizuje NAND
- Více kopií dat v NAND, aktualizace kopií později
- Lepší řešení: Bootloader v NOR, zbytek v NAND

U-Boot

- Velikostní limity SPL prakticky neexistují
- Plná podpora pro FS (ext234, reiserfs, vfat. . .)
- Podpora pro UBI/UBIFS pro NAND
- Podpora fitImage

Souhrn (1/4)

- HW musí mít korektní resetovací logiku
- Bootování z RAW NAND není dobrý nápad
- Nikdy neukládat nic důležitého přímo do RAW NAND
- Pokud nad RAW NAND nemáte UBI
  $\rightarrow$ Více kopií
  $\rightarrow$ ECC + Refresh později

Typy obrazů jádra

- zImage
  - Náchylný k poškození dat, nemusí být detekováno
  - Obsahuje pouze obraz jádra
  - Často používený
- uImage (legacy)
  - Slabý kontrolní součet CRC32
  - Obsahuje pouze obraz jádra
  - Často používený
- fitImage
  - Nastavitelný alg. pro kontrolní součet
  - Může obsahovat jádro, DTB, firmware...
  - Další vlastnosti...
  - Není rozšířen :-(

fitImage detailně

- Následník uImage
- Popisovač fitImage založen na DTS
- Nové vlastnosti lze snadno přidat
- Silnější kontrolní součty (SHA1, SHA256...)
- Může obsahovat více komponent (kernel, DTB, FW...)
- Může obsahovat více konfigurací
- U-Boot může verifikovat obsah oproti certifikátu

⇒ Ochrana před poškozením dat

⇒ Ochrana před cílenou změnou dat

- Linux kernel neumí generovat fitImage

uImage vs. fitImage: Vytvoření

```
/dts-v1/;
/ {
        description = "Linux kernel";
        #address-cells = <1>;
        images {
                kernel@1 {
                        description = "Linux kernel";
                        data = /incbin/("./arch/arm/boot/zImage");
                        arch = "arm";
                        os = "linux";
                        type = "kernel";
                        compression = "none";
                        load = <0x8000>;
                        entry = <0x8000>;
                        hash@1 {
                                algo = "crc32";
                        };
                };
        };
        configurations {
                default = "conf@1";
                conf@1 {
                        description = "Boot Linux kernel";
                        kernel = "kernel@1";
                        hash@1 {
                                algo = "crc32";
                        };
                };
        };
};
```

```
$ mkimage -f fit-image.its fitImage
```

```
$ mkimage -A arm -O linux -T kernel -C none -a 0x8000 -e 0x8000 -n "Linux kernel"
          -d arch/arm/boot/zImage uImage
```

uImage vs. fitImage: Boot

```
uImage    => load mmc 0:1 ${loadaddr} uImage
uImage    => bootm ${loadaddr}

fitImage => load mmc 0:1 ${loadaddr} fitImage
fitImage => bootm ${loadaddr}
```

- uImage je snadnější na sestavení
- uImage nepotřebuje soubor fit-image.its
- uImage a fitImage se startují stejně

uImage zatím vede. . .

uImage vs. fitImage: Device Tree Blob

```
...
/ {
        images {
                ...
+               fdt@1 {
+                       description = "Flattened Device Tree blob";
+                       data = /incbin/("./arch/arm/boot/dts/socfpga_cyclone5_mcvevk.dtb");
+                       type = "flat_dt";
+                       arch = "arm";
+                       compression = "none";
+                       hash@1 {
+                               algo = "sha256";
+                       };
+               };
                ...
        };
        configurations {
                conf@1 {
                        ...
+                       fdt = "fdt@1";
                        ...
                };
        };
};
```

uImage vs. fitImage: Boot s DTB

```
uImage   => load mmc 0:1 ${loadaddr} uImage
uImage   => load mmc 0:1 ${fdtaddr} socfpga_cyclone5_mcvevk.dtb
uImage   => bootm ${loadaddr} - ${fdtaddr}

fitImage => load mmc 0:1 ${loadaddr} fitImage
fitImage => bootm ${loadaddr}
```

- fitImage umožní update všech komponent najednou
- fitImage chrání DTB kontrolním součtem (zde sha256)
- Příkaz pro spuštení fitImage s DTB je stejný jako pro fitImage bez DTB

fitImage: Více konfigurací

```
...
/ {
        images {
                kernel@1 {};
                kernel@2 {};
                fdt@1 {};
                fdt@2 {};
                ...
        };
        configurations {
                conf@1 {
                        kernel = "kernel@1";
                        fdt = "fdt@1";
                        ...
                };
                conf@2 {
                        kernel = "kernel@1";
                        fdt = "fdt@2";
                        ...
                };
        };
};

=> bootm ${loadaddr}#conf@2
=> bootm ${loadaddr}:kernel@2
```

- fitImage podporuje předdefinované konfigurace
- Spuštění konfigurace pomocí znaku `#` (HASH)
- Spuštění komponentu fitImage pomocí znaku `:` (COLON)

fitImage: Ochrana FW blobů

```
...
/ {
        images {
                ...
+               firmware@1 {
+                       description = "My FPGA firmware";
+                       data = /incbin/("./firmware.rbf");
+                       type = "firmware";
+                       arch = "arm";
+                       compression = "none";
+                       hash@1 {
+                               algo = "sha256";
+                       };
+               };
                ...
        };
};

=> imxtract ${loadaddr} firmware@1 ${fwaddr}
=> fpga load 0 ${fwaddr}
```

- fitImage podporuje libovolné množství FW blobů
- fitImage chrání vaše bloby proti poškození

fitImage: Vylistování obsahu image

```
=> iminfo ${loadaddr}

## Checking Image at 10000000 ...
   FIT image found
   FIT description: Linux kernel and FDT blob for mcvevk
   Created:         2014-09-22  15:37:52 UTC
    Image 0 (kernel@1)
     Description:  Linux kernel
     Created:      2014-09-22  15:37:52 UTC
     Type:         Kernel Image
     Compression:  uncompressed
     Data Start:   0x100000d8
     Data Size:    3363584 Bytes = 3.2 MiB
     Architecture: ARM
     OS:           Linux
     Load Address: 0x00008000
     Entry Point:  0x00008000
     Hash algo:    crc32
     Hash value:   5c7efdb5
    Image 1 (fdt@1)
     Description:  Flattened Device Tree blob
     Created:      2014-09-22  15:37:52 UTC
     Type:         Flat Device Tree
     ...
    Default Configuration: 'conf@1'
    Configuration 0 (conf@1)
     Description:  Boot Linux kernel with FDT blob
     Kernel:       kernel@1
     FDT:          fdt@1
## Checking hash(es) for FIT Image at 10000000 ...
   Hash(es) for Image 0 (kernel@1): crc32+
   Hash(es) for Image 1 (fdt@1): crc32+
```

Souhrn (2/4)

- fitImage je super (a to není vše)
- fitImage chraní všechny bootovací soubory
- fitImage umožňuje zabalit všechny soubory do jednoho
  $\Rightarrow$Update všech bootovacích souborů najednou
- fitImage překonává uImage ve flexibilitě a rozšiřitelnosti
- fitImage není tak náchylný k tichému poškození dat

fitImage: Podpora podepisování

- Ochrana proti záměrné změně
- fitImage (nebo jeho node) se podepíše
- U-Boot zkonotroluje podpis proti veřejnému klíči
- Certifikát musí být v read-only umístění
- SHA1/SHA256 + RSA2048 a SHA256 + RSA4096

fitImage: Implementace podepsaného image

- Zapnutí control FDT v U-Boot
- Generování kryptomateriálu (OpenSSL)
- Generování control FDT s veřejným klíčem
- Sestavení U-Boot s podporou verifikace podepsaného fitImage
- Aktualizace U-Boot a test...

fitImage: Úpravy U-Boot

Změny pro konfiguraci zařízení:

```
#define CONFIG_OF_LIBFDT
#define CONFIG_OF_CONTROL

#define CONFIG_FIT
#define CONFIG_FIT_SIGNATURE
#define CONFIG_RSA

/* The FDT blob is not part of the U-Boot binary */
#define CONFIG_OF_SEPARATE
/* The $fdtcontroladdr points to where the FDT blob is */
#define CONFIG_EXTRA_ENV_SETTINGS "fdtcontroladdr=0x4000\0"
```

- SPL musí načíst control FDT na 0x4000
- Volba `CONFIG_OF_EMBED` pro vestavěný control FDT
- Volba `CONFIG_FIT_VERBOSE` pro extra debug

! Nyní je potřeba znovu zkompilovat 'tools/' (`make tools`)

fitImage: Generování kryptomateriálu

- Kryptomateriál umístíme do `key_dir="/work/keys/"`
- Sdílené jméno klíčů `key_name="my_key"`

- Generování **privátního** podepisovacího klíče (RSA2048):

```
$ openssl genrsa -F4 -out \
  "${key_dir}"/"${key_name}".key 2048
```

- Generování **veřejného** klíče:

```
$ openssl req -batch -new -x509 \
  -key "${key_dir}"/"${key_name}".key \
  -out "${key_dir}"/"${key_name}".crt
```

fitImage: Control FDT

Příklad control FDT (u-boot.dts):

```
/dts-v1/;
/ {
        model = "Keys";
        compatible = "denx,mcvevk";

        signature {
                key-dev {
                        required = "conf";
                        algo = "sha256,rsa2048";
                        key-name-hint = "my_key";
                };
        };
};
```

- Hodnota `my_key` v `key-name-hint` musí být ${key_name}
- Control FDT stále neobsahuje veřejný klíč, dostaneme se k tomu!

fitImage: Změny ve fitImage

Ukázka přidání signature node do fitImage ITS (fit-image.its):

```
...
/ {
        ...
        configurations {
                conf@1 {
                        ...
+                       signature@1 {
+                               algo = "sha256,rsa2048";
+                               key-name-hint = "my_key";
+                       };
                        ...
                };
        };
};
```

- Hodnota `my_key` v `key-name-hint` musí být `${key_name}`

fitImage: Sestavení

- Sestavení control FDT pro U-Boot s místem na veřejný klíč:
  ```
  $ dtc -p 0x1000 u-boot.dts -O dtb -o u-boot.dtb
  ```
- Sestavení fitImage s místem pro podpis:
  ```
  $ mkimage -D "-I dts -O dtb -p 2000" \
    -f fit-image.its fitImage
  ```
- Podepsání fitImage a přidání klíče do u-boot.dtb:
  ```
  $ mkimage -D "-I dts -O dtb -p 2000" -F \
    -k "${key_dir}" -K u-boot.dtb -r fitImage
  ```

- Az nyní je potřeba sestavit U-Boot, aktualizovat U-Boot a `u-boot.dtb` na zařízení a ověřit, že U-Boot stále správně startuje.

fitImage: Testování

Načíst podepsaný fitImage a zkusit pomocí `bootm start` (nebo `iminfo`):

- Úspěch (znak +):

```
Verifying Hash Integrity ...
sha256,rsa2048:my_key+ OK
```

- Selhání (znak -):

```
Verifying Hash Integrity ...
sha256,rsa2048:my_key- Failed to verify required
                       signature 'key-my_key'
```

- Podepisování dalších fitImage:

```
$ mkimage -D "-I dts -O dtb -p 2000" \
  -k "${key_dir}" -f fit-image.its -r fitImage
```

Souhrn (3/4)

- Podepsaný fitImage vypadá složitě na sestavení
- Celá komplikovaná procedura probíha pouze jednou
- Podepisování dalších fitImage je jeden příkaz

Načtení obrazu jádra

- Příkaz `load` pro vše co není NAND
- Příkaz `ubi*`/`ubifs*` pro NAND
- Neukládat data přímo do NAND

V Linuxu

- UBI/UBIFS na flash-based storage
- Ochrana dat pomocí IMA/EVM (obecně)

UBI/UBIFS

- UBI není kompletní řešení proti poškození dat
- UBI aktivně neobnovuje data na flash

⇒ K poškození dat může stále dojít!

⇒ Je potřeba "scrubber" (SLC NAND):

```
$ find / -exec cat {} > /dev/null 2>&1
```

! UBIFS a MLC NAND

Šifrování

- Šifrování U-Boot (pomocí BootROM)
- Šifrování U-Boot environmentu
  - U-Boot obsahuje `CONFIG_ENV_AES`
  - Nutno implementovat `env_aes_cbc_get_key`
- Šifrování obrazu jádra
  - U-Boot obsahuje `CONFIG_CMD_AES`
  - Použít `aes dec`
- Šifrování FS (pomocí `dm_crypt`)

Děkuji za pozornost

Kontakt: Marek Vasut <marex@denx.de>