fitImage & signature de paquets
Introduction

Mickaël Tansorier

Présentation du fonctionnement des fitImage et
du fonctionnement de signature de paquets

Plan

Images Linux

- **Image** : image générique binaire
- **zImage** : image générique binaire compressé
- **uImage** : image avec une entête d'information utilisé par U-Boot
- **fitImage** : enveloppe d'image pouvant contenir plusieurs noyaux, devicetree, firmware. Chaque image peut être signé, et d'autres choses

Common kernel image types

- zImage
  - Prone to silent data corruption, which can go unnoticed
  - Contains only kernel image
  - In widespread use
- uImage (legacy)
  - Weak CRC32 checksum
  - Contains only kernel image
  - In widespread use
- fitImage
  - Configurable checksum algorithm
  - Can be signed
  - Contains arbitrary payloads (kernel, DTB, firmware...)
  - There is more !
  - Not used much :-(

The fitImage in detail

- Successor to uImage
- Descriptor of image contents based on DTS
- Can contain multiple files (kernels, DTBs, firmwares...)
- Can contain multiple configurations (combo logic)
- New image features can be added as needed
- Supports stronger csums (SHA1, SHA256...)

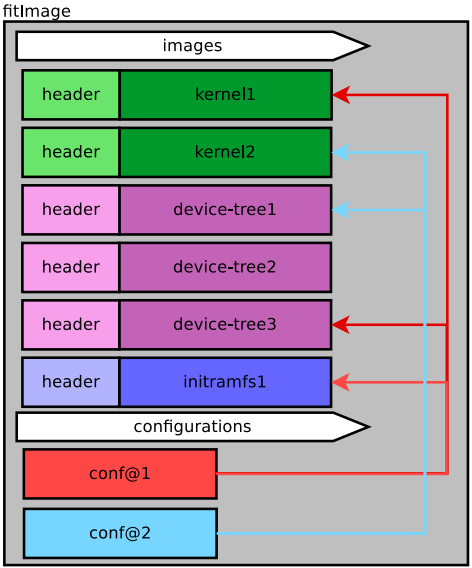
⇒ Protection against silent corruption

- U-Boot can verify fitImage signature against public key

⇒ Protection against tampering

- Linux build system can not generate fitImage :-(
- Yocto can not generate fitImage **yet** :-)

What's a fitImage?



- several talks given to present the fitImage, the reasons behind and the challenges
- it's basically a container for multiple binaries with hashing and signature support
- it also supports forcing a few binaries to be loaded together,
- supports different architectures, OSes, image types, ... => can be found in `common/image.c`

```
KERNEL=/path/to/zImage
KEYNAME=my_key
```

```
/dts-v1/;

/ {
    description = "fitImage for sign Kernel image
        and DTB";
    #address-cells = <1>;

    images {
        kernel@1 {
            description = "Linux Kenel";
            data = /incbin/("%KERNEL%");
            type = "kernel";
            arch = "arm";
            os = "linux";
            compression = "none";
            load = <0x12000000>;
            entry = <0x12000000>;
            signature@1 {
                algo = "sha256,rsa4096";
                key-name-hint = "%KEYNAME%";
            };
        };
        fdt@1 {
            description = "Devicetree";
            data = /incbin/("%DTB%");
            type = "flat_dt";
            arch = "arm";
            compression = "none";
            load = <0x18000000>;
            entry = <0x18000000>;
            signature@1 {
                algo = "sha256,rsa4096";
                key-name-hint = "%KEYNAME%";
            };
        };
    };
    configurations {
        default = "conf@1";
        conf@1 {
            kernel = "kernel@1";
            fdt = "fdt@1";
        };
    };
};
```

Plusieurs type de signature sont disponible : `common/image-sig.c`

```
struct image_sig_algo image_sig_algos[] = {
    {
        "sha1,rsa2048",
        rsa_sign,
        rsa_add_verify_data,
        rsa_verify,
        &checksum_algos[0],
    },
    {
        "sha256,rsa2048",
        rsa_sign,
        rsa_add_verify_data,
        rsa_verify,
        &checksum_algos[1],
    },
    {
        "sha256,rsa4096",
        rsa_sign,
        rsa_add_verify_data,
        rsa_verify,
        &checksum_algos[2],
    }

};
```

Crée un devicetree spécifique

```
/dts-v1/;
/ {
    model = "Keys";
    compatible ="vendor,board";
    signature {
        key-%KEYNAME% {
            required = "image";
            algo = "sha256,rsa4096";
            key-name-hint = "%KEYNAME%";
        };
    };
};
```

Pour le générer le dtb

```
dtc -p 4096 $(@D)/u-boot_pubkey.dts -O dtb -o $(@D)/u-boot_pubkey.dtb
```

Ce qui donne :

```
$ cat u-boot_pubkey.dtb
vendor,board signature key-my_key image sha256,rsa4096 my_key
    modelcompatiblerequiredalgokey-name-hint
```

Pour y ajouter la clé public

```
mkimage -D "-I dts -O dtb -p 4096" -f $(@D)/fitImage.its -K $(@D)/u-
    boot_pubkey.dtb -k $(@D) -r fitImage
```

Ce qui donne :

```
$ cat u-boot_pubkey.dtb
vendor,board signature key-my_key
[...]
image sha256,rsa4096 my_key modelcompatiblerequiredalgokey-name-
    hintrsa,num-bitrsa,n0-inversersa,exponentrsa,modulusra,r-
    squaredsquared
```

Pour s'assurer que le device tree contenant la clé public soit dans dans le binaire u-boot, il faut que cette option soit à non :

```
BR2_TARGET_UBOOT_USE_CUSTOM_CONFIG
```

Pour ajouter dans notre U-boot alors que l'on a pas de dtb, on utiliser l'option EXT_DTB de make :

```
make CROSS_COMPILE=arm-linux-gnueabihf- EXT_DTB=u-boot_pubkey.dtb
```

Ok maintenant il faut siqner le kernel linux.
Il faut le signer avant de compiler u-boot puisque pour ajouter la clé public au devicetree il faut executer mkimge avec en entrée l'its et en sortier le fitImage :

```
mkimage -D "-I dts -O dtb -p 4096" -f $(@D)/fitImage.its -K $(@D)/u-
    boot_pubkey.dtb -k $(@D) -r fitImage
```

Le noyau Linux est bien construit avant U-boot, donc pas besoin d'ajouter de dépendance.

J'ai rajouté quelques config dans buildroot.

```
BR2_PACKAGE_UBOOT_TOOLS_FIT_SUPPORT=y
BR2_PACKAGE_UBOOT_TOOLS_FIT_SIGNATURE_SUPPORT=y
BR2_PACKAGE_HOST_UBOOT_TOOLS=y
BR2_PACKAGE_HOST_UBOOT_TOOLS_FIT_SUPPORT=y
BR2_PACKAGE_HOST_UBOOT_TOOLS_FIT_SIGNATURE_SUPPORT=y

BR2_TARGET_UBOOT_NEEDS_OPENSSL=y

BR2_TARGET_UBOOT_SIGN_FITIMAGE=y
BR2_TARGET_UBOOT_ITS="$(CONFIG_DIR)/board/eolane/modx6/fitImage.its"
BR2_TARGET_UBOOT_SIGN_DTS="$(CONFIG_DIR)/board/eolane/modx6/u-
    boot_pubkey.dts"
BR2_TARGET_UBOOT_KEY_NAME="my_key"
BR2_TARGET_UBOOT_KEY_SERVER="bep@hgweb:/home/apache/distribution/keys/
    "
```

Vérification de la signature

il est possible de tester la signature d'une image avec :

```
fit_check_sign -f output/images/fitImage -k u-boot_pubkey.dtb
```

dans `./output/build/host-uboot-tools-2017.07/tools/fit_check_sign`

Documentation :

- https://elinux.org/images/e/e0/Josserand-schulz-secure-boot.pdf
- https://www.denx.de/wiki/pub/U-Boot/Documentation/multi_image_booting_scenarios.pdf
- https://elinux.org/images/8/8a/Vasut--secure_and_flexible_boot_with_u-boot_bootloader.pdf

Il faut une clé paire de clé :

```
$ openssl genrsa -out my_key.key 4096
$ openssl req -batch -new -x509 -key my_key.key -out my_key.crt
```

Pour extraire le clé publique du certificat

```
openssl x509 -pubkey -noout -in my_key.crt > my_key.pub
```

Pour signer :

```
openssl dgst -sha256 -sign my_key.key -out file_to_sign.sha256
    file_to_sign
```

Si la signature est sous forme binaire, on peut la convertir sous forme textuel :

```
openssl base64 -in file_to_sign.sha256 -out file_to_sign.sign
```

Pour signer :

```
openssl dgst -sha256 -verify my_key.pub -signature file_to_sign.sha256
    file_to_sign
```

Si la signature est sous forme binaire, on peut la convertir sous forme textuel :

```
openssl base64 -d -in file_to_sign.sign -out file_to_sign.sha256
```

Question ?

Enfin je vais essayer de répondre...