Introduction

Das U-Boot typically is employed as a second-stage boot loader responsible for loading the Linux operating system's kernel and related images and passing control on to the OS. The boot loader is also responsible of verifying the integrity and authenticity of the loaded images to ensure only authentic software is allowed to run. U-Boot's verified boot feature is used to achieve this. This method only applies to the flattened image tree (FIT) image format.

The FIT image is based on a previously developed device tree format used in both U-Boot and the Linux kernel to store and pass configuration information. An example image tree source (the text format used to generate FIT images) from the documentation is reproduced below to illustrate the concept:

```
/ {
    images {
        kernel-1 {
            data = <data for kernel1>
            hash-1 {
                algo = "sha1";
                value = <...kernel hash 1...>
            };
        };
        kernel-2 {
            data = <data for kernel2>
            hash-1 {
                algo = "sha1";
                value = <...kernel hash 2...>
            };
        };
        fdt-1 {
            data = <data for fdt1>;
            hash-1 {
                algo = "sha1";
                value = <...fdt hash 1...>
            };
        };
        fdt-2 {
            data = <data for fdt2>;
            hash-1 {
                algo = "sha1";
                value = <...fdt hash 2...>
            };
        };
    };
    configurations {
        default = "conf-1";
        conf-1 {
            kernel = "kernel-1";
            fdt = "fdt-1";
            signature-1 {
                algo = "sha1,rsa2048";
                value = <...conf 1 signature...>;
            };
        };
        conf-2 {
            kernel = "kernel-2";
            fdt = "fdt-2";
            signature-1 {
                algo = "sha1,rsa2048";
                value = <...conf 1 signature...>;
            };
        };
    };
};
```

This image tree source describes two kernels with two flattened device tree blobs (FDTs) as well as two configurations denoting which images are to be used together. The complete configuration may also include other images e.g. RAM disks. Hash values for individual images are computed during image build.

When an image is signed using the **mkimage** tool provided with U-Boot, the image is modified by adding several properties to corresponding signature nodes, specifically **hashed-strings**, **hashed-nodes**, **timestamp**, **signer-version**, **signer-name**, and **value**. The **value** property holds the actual signature, while the **hashed-strings** and **hashed-nodes** properties represent the elements that were used to compute the signed hash value. For example:

```
conf@1 {
  description = [REMOVED];
  kernel = "kernel@1";
  fdt = "fdt@1";
  ramdisk = "ramdisk@1";
  signature@1 {
    hashed-strings = [00 00 00 00 00 00 00 8e];
    hashed-nodes = "/", "/configurations/conf@1", "/images/fdt@1",
"/images/fdt@1/hash@1", "/images/kernel@1", "/images/kernel@1/hash@1",
"/images/ramdisk@1", "/images/ramdisk@1/hash@1";
    timestamp = [5d cb 49 dc];
    signer-version = "2019.07";
    signer-name = "mkimage";
    value = [03 d9 d7 e8 5a cf ..];
    algo = "sha256,rsa4096";
    key-name-hint = [REMOVED];
    sign-images = "fdt", "kernel", "ramdisk";
  };
};
```

The string block data starting from offset 0 to offset 0x8E was included in the hash computation along with enumarated tree nodes, which produces the given RSA signature.

Description

It was found that U-Boot does not verify the contents of **hashed-nodes** correlate with the sub-images required to be loaded by the configuration, specified e.g. in the **kernel**, **fdt**, and **ramdisk** configuration properties. This allows us to craft another configuration with the same signature node but referencing a different sub-image(s):

```
conf@2 {
  description = "Super 1337 Configuration";
  kernel = "kernel@2";
  fdt = "fdt@1";
  ramdisk = "ramdisk@1";
  signature@1 {
    hashed-strings = [00 00 00 00 00 00 00 8e];
    hashed-nodes = "/", "/configurations/conf@1", "/images/fdt@1",
"/images/fdt@1/hash@1", "/images/kernel@1", "/images/kernel@1/hash@1",
"/images/ramdisk@1", "/images/ramdisk@1/hash@1";
    timestamp = [5d cb 49 dc];
    signer-version = "2019.07";
    signer-name = "mkimage";
    value = [03 d9 d7 e8 5a cf ..];
    algo = "sha256,rsa4096";
    key-name-hint = [REMOVED];
    sign-images = "fdt", "kernel", "ramdisk";
  };
};
```

As by design only certain parts of the FIT image are hashed, it is possible to insert arbitrary FIT nodes after configurations have been signed without invalidating any signatures, adding both arbitrary configurations as well as arbitrary sub-images. Note that this fact is explicitly documented. It is also possible to change the default property of the configurations node to suggest the crafted configuration to be chosen.

Impact

An attacker having a properly signed FIT image is able to craft arbitrary FIT images that would pass signature validation, resulting in booting and execution of untrusted code.

The exploitation relies on the fact that the crafted configuration will be chosen to be booted. This may occur, for example, when the attacker is able to modify the **default** property of the **configurations** node and the setup does not explicitly choose to boot a specific configuration. Consequently, one way of mitigating the issue is to explicitly specify the configuration name as part of the **bootm** command arguments, for example:

```
bootm ${loadaddr}#conf@1 - ${fdtaddr}
```

Affected versions

U-Boot versions 2018.03 and 2020.01 were verified to be affected. Versions prior to 2018.03 may be affected as well.

Solution

Apply patches or update to a fixed version when available.

Preliminary patches have been posted to the mailing list for review.