FIT happens

how Secure Boot wasn't



Ahmad Fatoum – a.fatoum@pengutronix.de



About Me

Ahmad Fatoum

Pengutronix

a3f

a.fatoum@pengutronix.de

@a3f@fosstodon.org



- Kernel and Bootloader Porting
- Driver and Graphics Development
- System Integration
- Embedded Linux Consulting

Overview

- FIT: History and Introduction
- FIT: Increase in Scope
- FIT: Signing
- FIT: Exploiting
- FIT: Fixing?
- FIT: Surviving

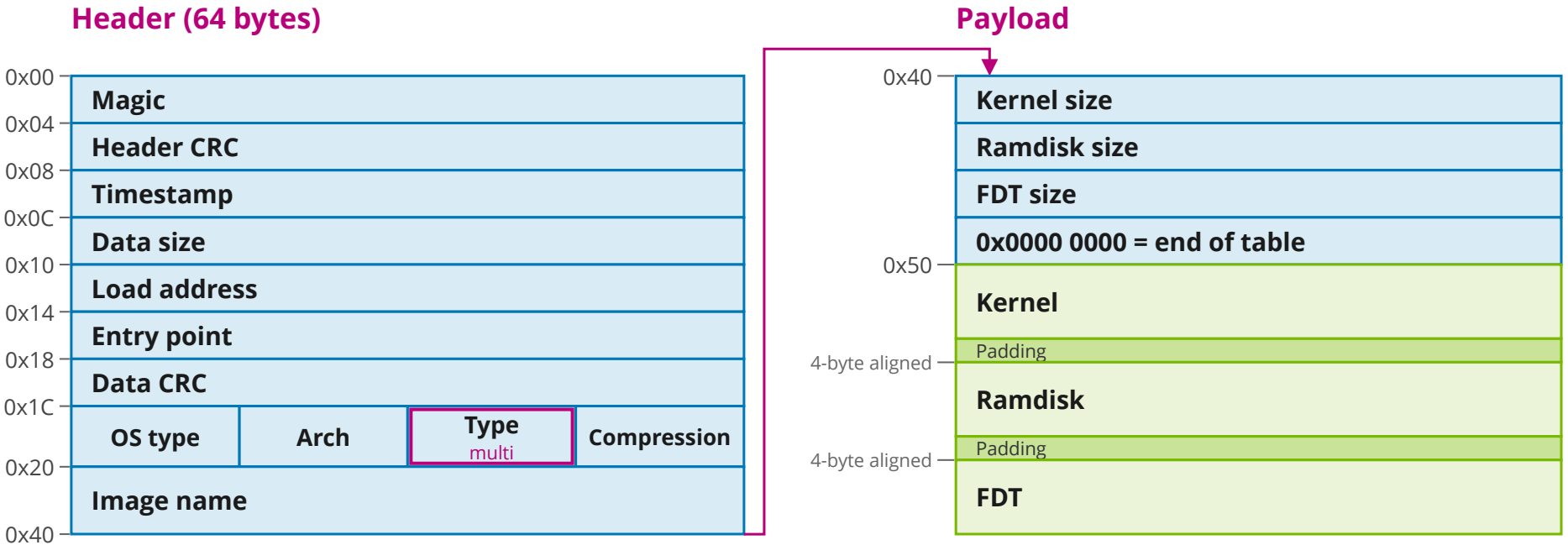


Getting FIT

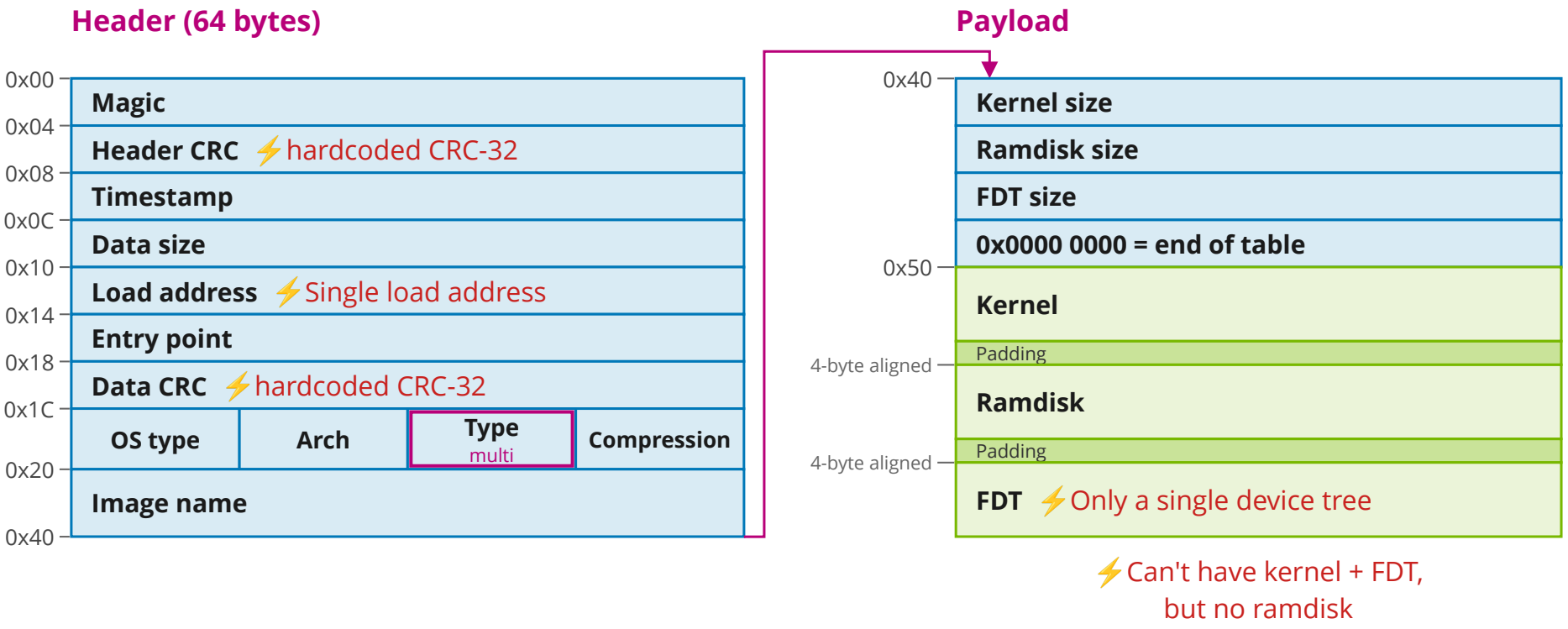
(a short history narrated by someone who wasn't even there)



uImage multi-file image

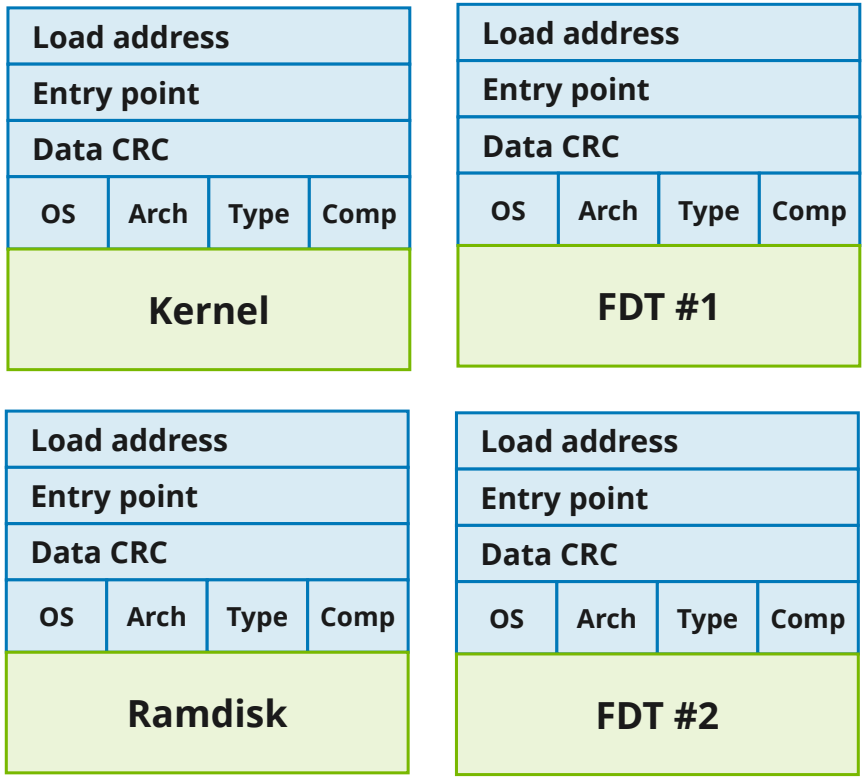


uImage multi-file image



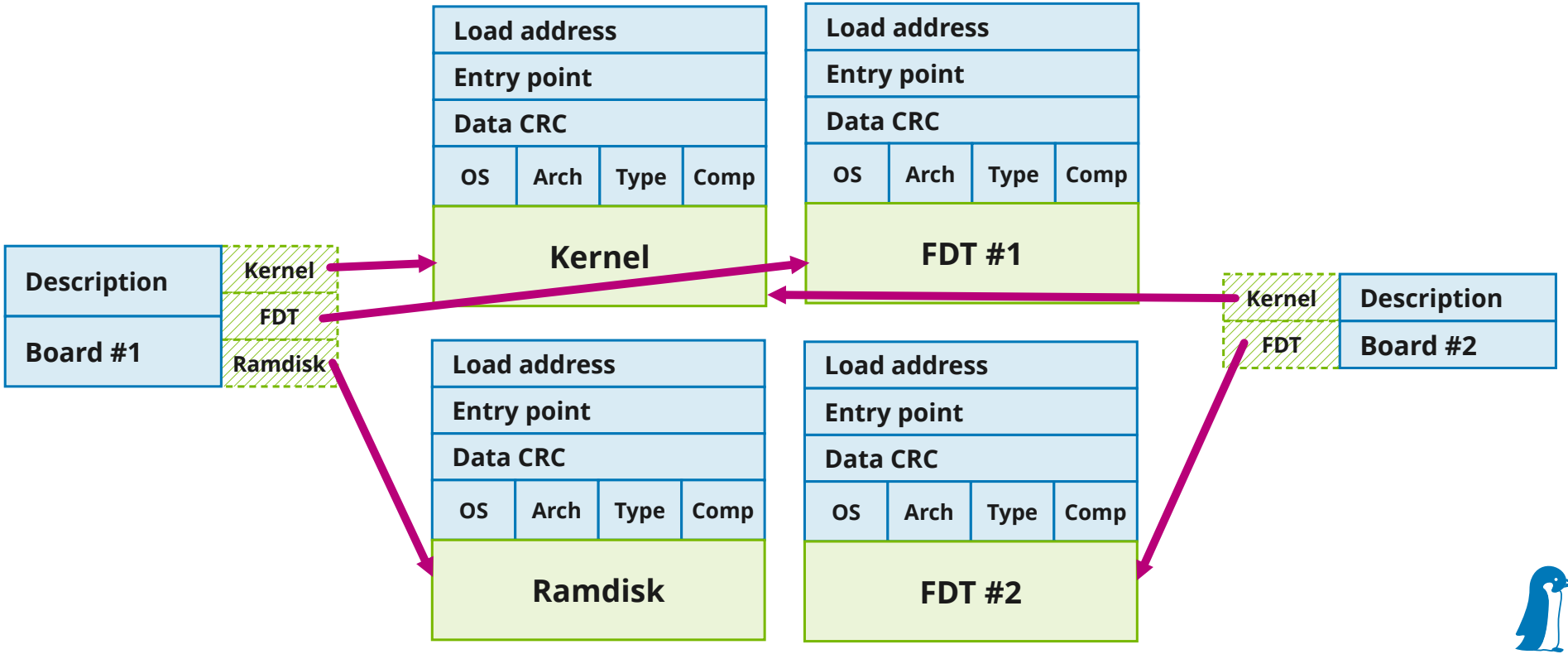
The need for *proper* multi file images

- Define multiple *images*, each with its own metadata



The need for *proper* multi file images

- Combine multiple images into bootable *configurations*

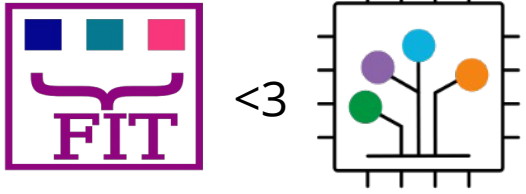


Device Tree as Container File Format

- Flexible
  - any number of key/value pairs
  - any number of nested nodes
- Tooling available
  - `dtc`, `libfdt`, `fdtdump` …
- Code size
  - U-Boot already has a parser!

```
/ {
    node@0 {
        property1;
        property2 = "value2";
        /* […] */
    };

    node-1 {
        property1;
        property3 = "value3";
    };
};
```



FIT: ITS

FIT

```
{
    description = "Linux-6.19.0-rc6";
    #address-cells = <0x1>;
    timestamp = <0x697a0898>;
    configurations {
        conf-11 {
             compatible = "fsl,imx8mm-ddr4-evk",
                          "fsl,imx8mm";
             description = "FSL i.MX8MM DDR4 EVK";
             kernel = "kernel";
             ramdisk = "ramdisk";
             fdt = "fdt-11";
         };
         conf-419 {
             compatible = "radxa,rock3a",
                          "rockchip,rk3568";
             description = "Radxa ROCK 3A";
             kernel = "kernel";
             ramdisk = "ramdisk";
             fdt = "fdt-419";
        };
    };
};
```

FIT

```
{ images {
    kernel {
        description = "Linux-6.19.0-rc6";
        type = "kernel_noload";
        arch = "arm64";
        os = "linux";
        compression = "gzip";
        data = /* 10281261 bytes omitted */;
        load = <0x0>;
        entry = <0x0>;
    };
    ramdisk {
        description = "Ramdisk";
        type = "ramdisk";
        arch = "arm64";
        compression = "none";
        os = "linux";
        data = /* 2020830 bytes omitted */;
    };
    fdt-11 {
        description = "imx8mm-ddr4-evk.dtb";
        type = "flat_dt";
        arch = "arm64";
        compression = "gzip";
        data = /* 9777 bytes omitted */;
    };
    fdt-419 {
        description = "rk3568-rock-3a.dtb";
        type = "flat_dt";
        arch = "arm64";
        compression = "gzip";
        data = /* 13423 bytes omitted */;
    };
} };
```



FIT: Matching

FIT ITS

```
{
    description = "Linux-6.19.0-rc6";
    #address-cells = <0x1>;
    timestamp = <0x697a0898>;
    configurations {
        conf-11 {
            compatible = "fsl,imx8mm-ddr4-evk",
                         "fsl,imx8mm";
            description = "FSL i.MX8MM DDR4 EVK";
            kernel = "kernel";
            ramdisk = "ramdisk";
            fdt = "fdt-11";
        };
        conf-419 {
            compatible = "radxa,rock3a",
                         "rockchip,rk3568";
            description = "Radxa ROCK 3A";
            kernel = "kernel";
            ramdisk = "ramdisk";
            fdt = "fdt-419";
        };
    };
};
```

Bootloader DT

```
/ {
    compatible = "radxa,rock3a",
                 "rockchip,rk3568";

    /* ~6700 more lines omitted */
};
```

✅ **Match!**

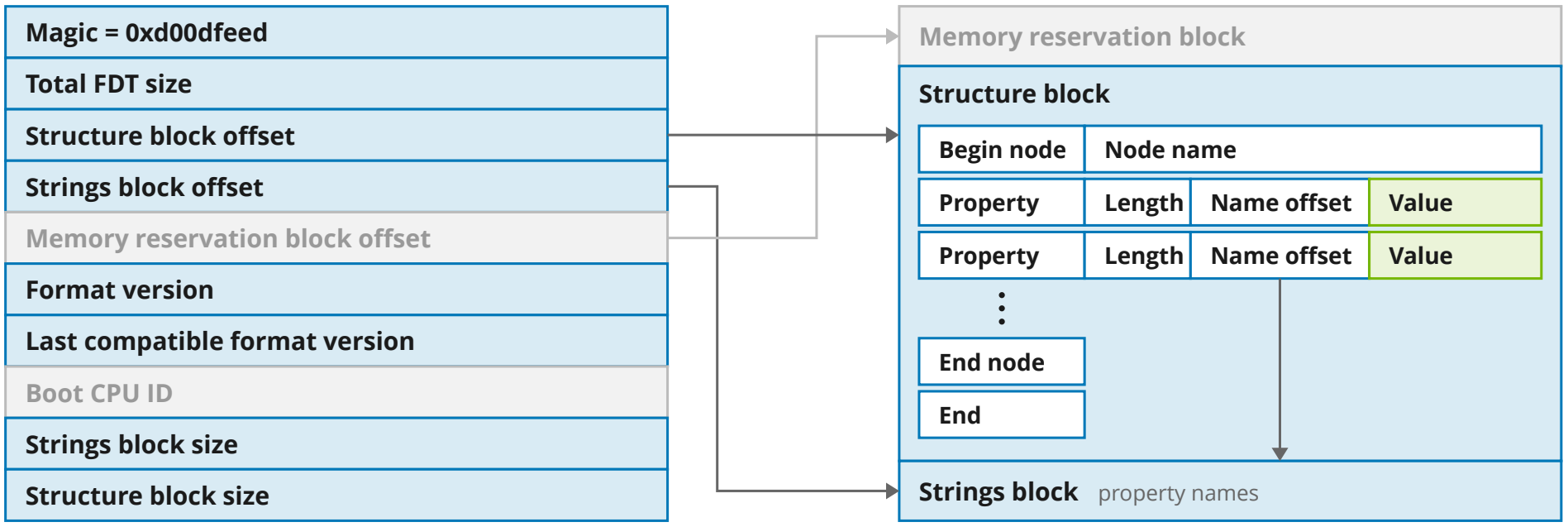
This Image Tree Source was written by Linux's `make image.fit` (see FOSDEM 2026), but there are various other ways:

- By hand
- By build system wrappers (kernel-fit-image.bbclass, fitimage.bbclass)
- By `mkimage -f auto*`



FIT Binary Format

Header (40 bytes)



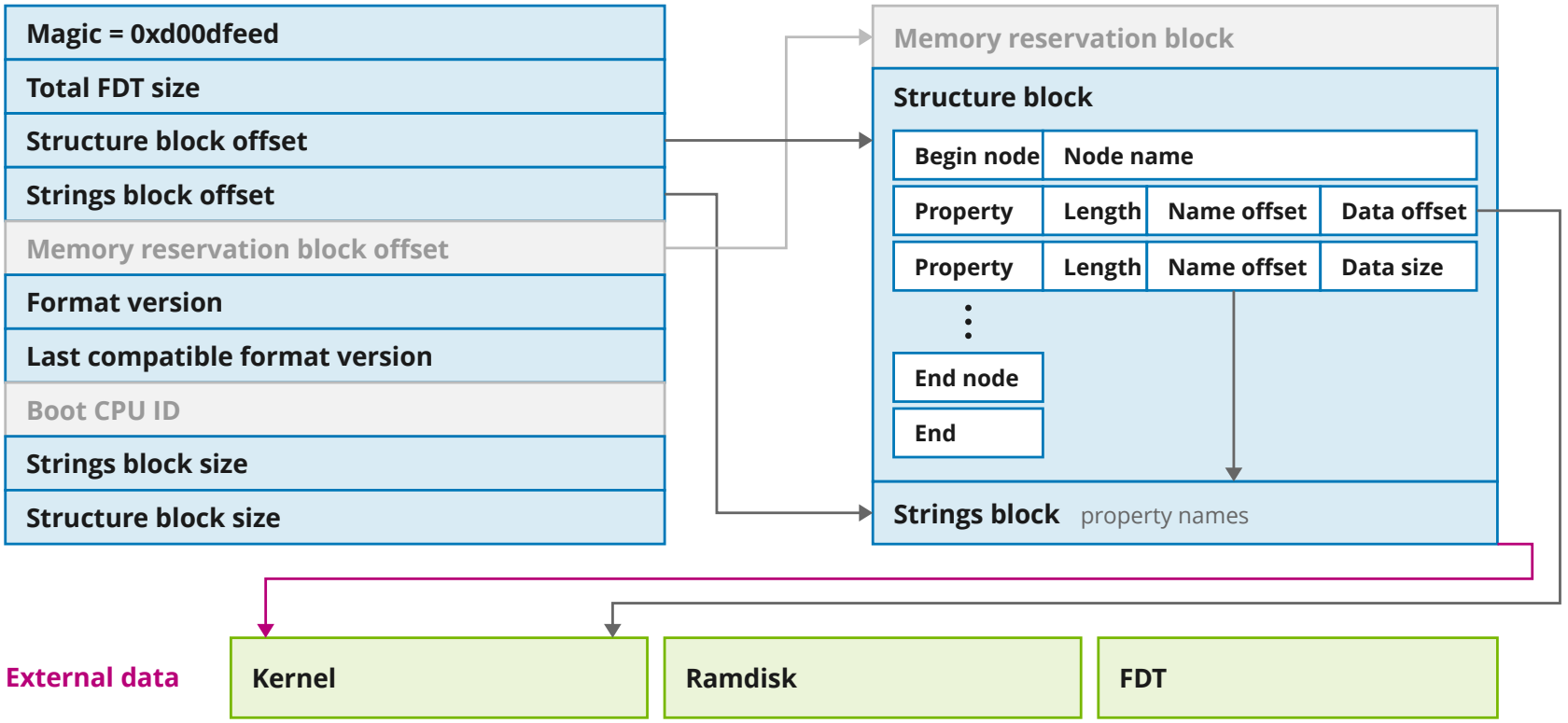
- FIT detection not possible without *walking* the Structure block, e.g. to check existence of `/configurations`

One Size FITs all

(How we may have lost the light)



FIT as a partition table



- FIT size unknown unless all image nodes are parsed!

Putting U-Boot into the FIT

- U-Boot SPL loads the next stages from a FIT: `u-boot.itb`
- Parsed by SPL at highest available privilege level

from `arch/arm/dts/imx8mm-u-boot.dtsi`

```
configurations {
        default = "@config-DEFAULT-SEQ";
        @config-SEQ {
                description = "NAME";
                fdt = "fdt-SEQ";
                firmware = "uboot";
#ifndef CONFIG_ARMV8_PSCI
#ifdef CONFIG_OPTEE
                loadables = "atf", "tee";
#else
                loadables = "atf";
#endif
#endif
        };
};
```

Unfun Fact:
Pre-checks for FDT are omitted from SPL for size reasons, although it's more security-sensitive!

Pet peeve: u-boot.itb assembled outside U-Boot (`imx-mkimage`, `uboot-sign.bbclass`): get it wrong, fix it everywhere



Certified FIT

(Shoehorning signing into everybody's favorite boot format)



Signing Images: What's signed

```
/images { kernel-1 {
    data  = [ … ];   /* the ONLY signed bytes */
    type        = "kernel";
    arch        = "arm64";
    os          = "linux";
    compression = "none";
    load        = <0x40080000>; /* attacker-chosen */
    entry       = <0x40080000>; /* attacker-chosen */
    signature-1 { algo; key-name-hint; value }
}};

/configurations { default; conf-1 { kernel; fdt; } }
/ { description; timestamp; }
```

⚡ The signature only covers data!



Fundamentally insecure

- Just place your unsigned payload anywhere and have `entry` point at it
- U-Boot does no bounds checking on `entry`
  - Wouldn't help anyway: a complete kernel image has no shortage of gadgets for ROP
- barebox bootm never accepted image signatures,
  as they would allow mix-and-match
- See Rasmus Villemoes' PoC on a BeagleBone Black, u-boot@, 2026-03-05

Tangent: Would W^X have helped here?

1) A year after CONFIG_MMU_PGPROT landed in U-Boot, only two platforms select it
2) The U-Boot heap is still executable regardless, whereas barebox does mark the heap non-executable
3) Does usually not matter either way: ARM kernel boot without EFI is with MMU off…



Who does Image Signing?

- OE-core `uboot-sign.bbclass` with `SPL_SIGN_ENABLE = "1"`
  - TF-A, OP-TEE, FDT and U-Boot are signed one by one: `entry` is not signed
  - Patch fixing it is under discussion:
    - uboot-sign: sign SPL FIT configurations instead of images
- `mkimage -f auto -g … -o …` signs images, not configurations
- Effectively all upstream U-Boot SPLs that implement `board_fit_image_post_process()`?!

Note: `os = "linux"` on ARM64 ignores `entry`. One workaround is to patch `os = "vxworks"` alongside the `entry` change



Signing Configurations

```
images {
        kernel-1 {
                description = "Linux kernel";
                data = /incbin/("./zImage");
                compression = "none";

                hash-1 {
                        value = [ … ];
                        algo = "sha256";
                };
        };
};
```

- Hash still covers only the data, but…



Signing Configurations

```
conf-1 {
        kernel = "kernel-1";
        signature-1 {
                algo = "sha256,rsa2048";

                /*  input "parameter" for mkimage */
                sign-images = "kernel";

                /* output "parameters" for mkimage */
                hashed-nodes = "/", "/configurations/conf-1",
                               "/images/kernel-1",
                               "/images/kernel-1/hash-1";
                hashed-strings = <0x00 0x82>;
                value = [ … ];
        };
};
```

- The signature covers the listed nodes, including the image hashes



So what can't you do, and what can you?

Breaks the signature:

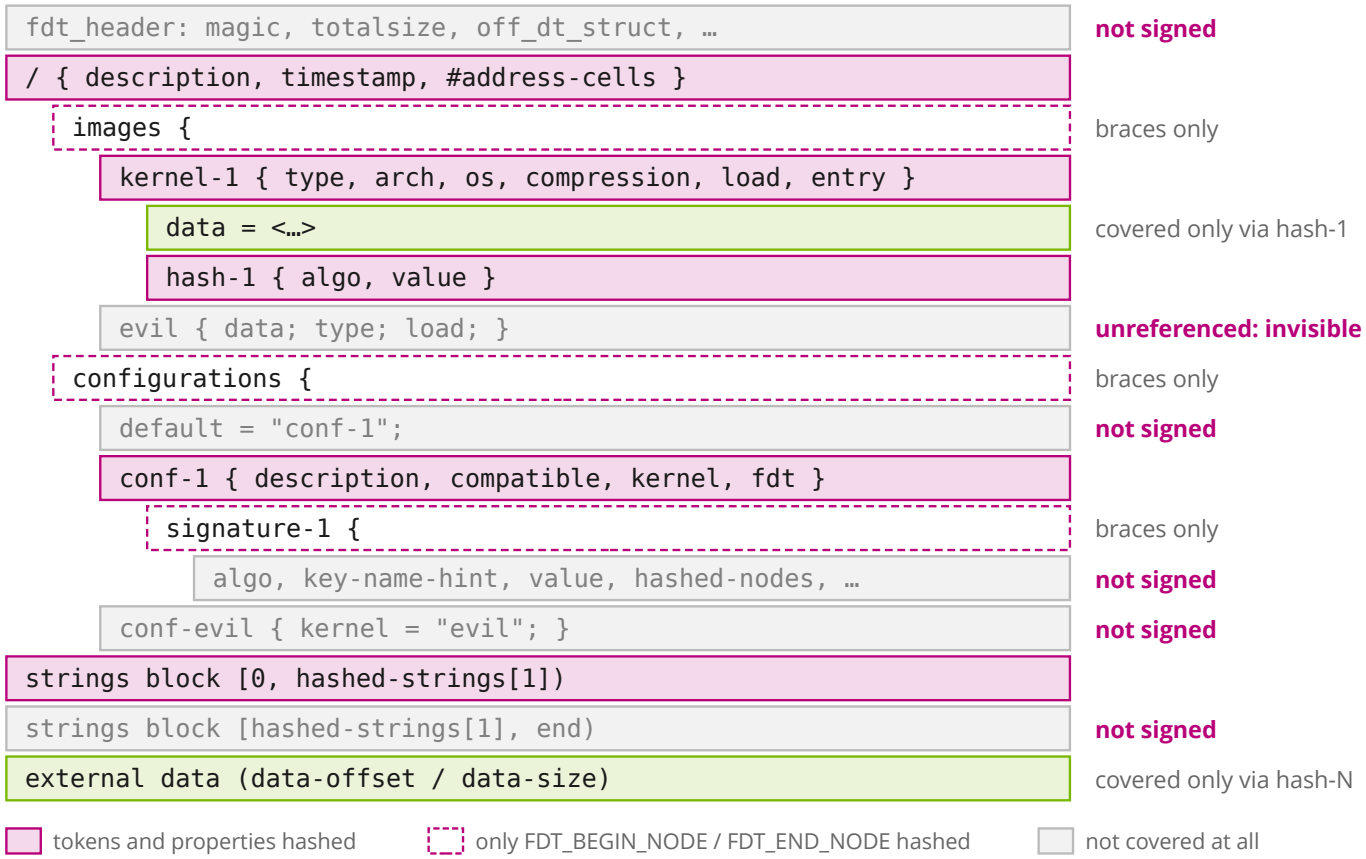
- Change the **`entry`** address of `/images/kernel-1`
- Add a **new property** to `/` or **`/configurations`**`/conf-1`
- Change **`algo`** in `/images/kernel-1/hash-1`
- Add a **subnode** to `/images/kernel-1`
- Add a new **top-level node**

Still verifies:

- Add complete **extra images** after the fact
- Add complete **extra configurations** after the fact
- Change `/configurations/`**`default`**
- Add a **property** to **`/images`**
- Change *some* **properties in the signature node**!



Signing Configurations: What's signed



C Bingo Card: Everything is there

	barebox	**U-Boot**
integer overflow	`7cf25e0733f0 198054e168f8 870a0ef748ef f936c591c269 26136fd068d7`	`8642b2178d2c 01460dfbda16 3158b844335a 491c7b6f42a8`
unbounded memcpy / buffer overflow	`96628249f26e 38ee3bd662a8`	`f1c85688ab13 21ebf2adde3e ca774b94d663 8f71d7170f5c a84c7670a698`
deep recursion	`05b6373a0fda (DoS)`	`9e0d2fb42965 (Stack Overflow)`
OOB read, string table & friends	`8a6b7db572c7 002bb5e97fa9 4e9331dd33be 7a3cb7e6fd63 b578ab6d4df9 ef2e9ab5611c 6722f7f61ed2 f5d1df1d1535`	`e73328612534 69f6272b24a2 79f8f31d58df 84e250c0a85a`
off-by-one	`e80ce6356246`	`4bc7deee9095`
double free / UAF	`3d6d2414d85e ba345a71e85e`	
OOB write (memset)		`0a10b49206a2`
NULL deref	`2b8bf0ce90ca 6553d2c2a5e3`	`f3d2ff3f5c3f 7304d569e615 2a3be302d5dd 952295660520`

Subset of fixes to FIT and its dependencnies



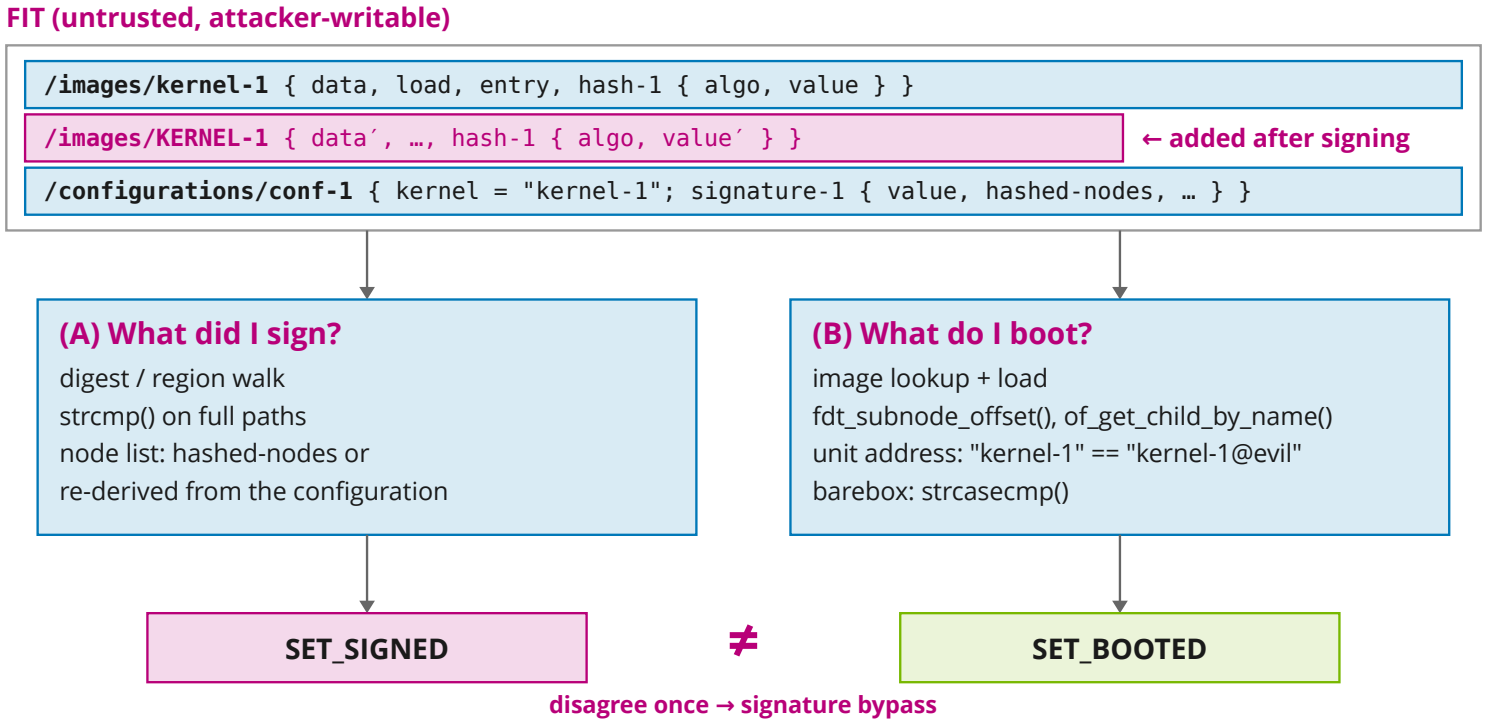
FIT happens

(How Secure Boot wasn't)



FIT is tricky to parse

- The FDT is parsed before the signature is verified
- Goal: Get FIT consumers to confuse nodes



Get FIT consumers to confuse nodes

- Via memory safety issues
  - FDT use is widespread, but except for FIT, it's always *trusted* input
- Via fuzziness in the APIs:
  - unit addresses (@)
  - case sensitivity
- Via tricking walkers:
  - slashes in node names
  - non-NUL-terminated strings
- Via abusing necessary untrusted data:
  - `default`
  - `hashed-nodes`
  - …



CVE-2026-33243 / CVE-2026-46728

```
/images/images/kernel-1 { data = GOOD; hash-1 { value = H(GOOD) } }  ← hashed
/images/kernel-1        { data = EVIL; hash-1 { value = H(EVIL) } }  ← booted

hashed-nodes = "/", "/configurations/conf-1", "/images/images/kernel-1", …
                                              ↑ rewritten by the attacker
```

- Reported by Apple Security Engineering and Architecture
- `hashed-nodes` tells the verifier which nodes were hashed, but is not hashed itself
- Fix: rebuild the node list from the configuration's actual image references
  - Fixed in U-Boot >= v2026.04, barebox >= v2026.03.1, FIT Spec commit `1776470fa1c8`
- Details in advisory: GHSA-3fvj-q26p-j6h4
- Welcome side-effect: sign-images also on the chopping block now: spec PR #55, mkimage commit `f252d228be2bf252d228be2b`



Show of Hands

If you

- Have a verified boot project using FIT
- and have not heard of this particular issue

Please raise your hand ✋



Trimming the FIT

(A hash was meant to be computed sequentially, dammit)



FIT keeps on growing

- Overlapping images: `image-data` lets images share one payload
- FIT as partition table #2: dm-verity edition
- Open PRs:
  - PKCS#7
  - Measured Boot
  - Logos and os-release
  - Revocation



ELC-E 2025: "What I look forward to"

- In barebox:
  - Security Policies hopefully part of barebox v2025.09.0 or v2025.10.0
    - Upstreamed external build-system integration: devtool or bitbake
  - Remaining fuzzer memory leaks fixed and application for oss-fuzz
  - Upstream downgrade protection
- General wishlist:
  - More hardening and fuzzing for boot loaders and firmware
  - More documentation about trust expectations for non-experts
  - FIT Images: Verify full image *before* parsing?
  - Rewriting critical components in memory-safe languages

Questions?



ELC-E 2025: "What I look forward to"

- In barebox:
  - Security Policies hopefully part of barebox v2025.09.0 or v2025.10.0 ✅
    - Upstreamed external build-system integration: devtool or bitbake ⚙️
  - Remaining fuzzer memory leaks fixed and application for oss-fuzz ✅ 13 barebox tests in oss-fuzz now!
  - Upstream downgrade protection
- General wishlist:
  - More hardening and fuzzing for boot loaders and firmware ✅ 4 U-Boot tests in oss-fuzz now!
  - More documentation about trust expectations for non-experts ✅, barebox Threat Model
  - FIT Images: Verify full image *before* parsing?
  - Rewriting critical components in memory-safe languages ⚙️

~~Questions?~~

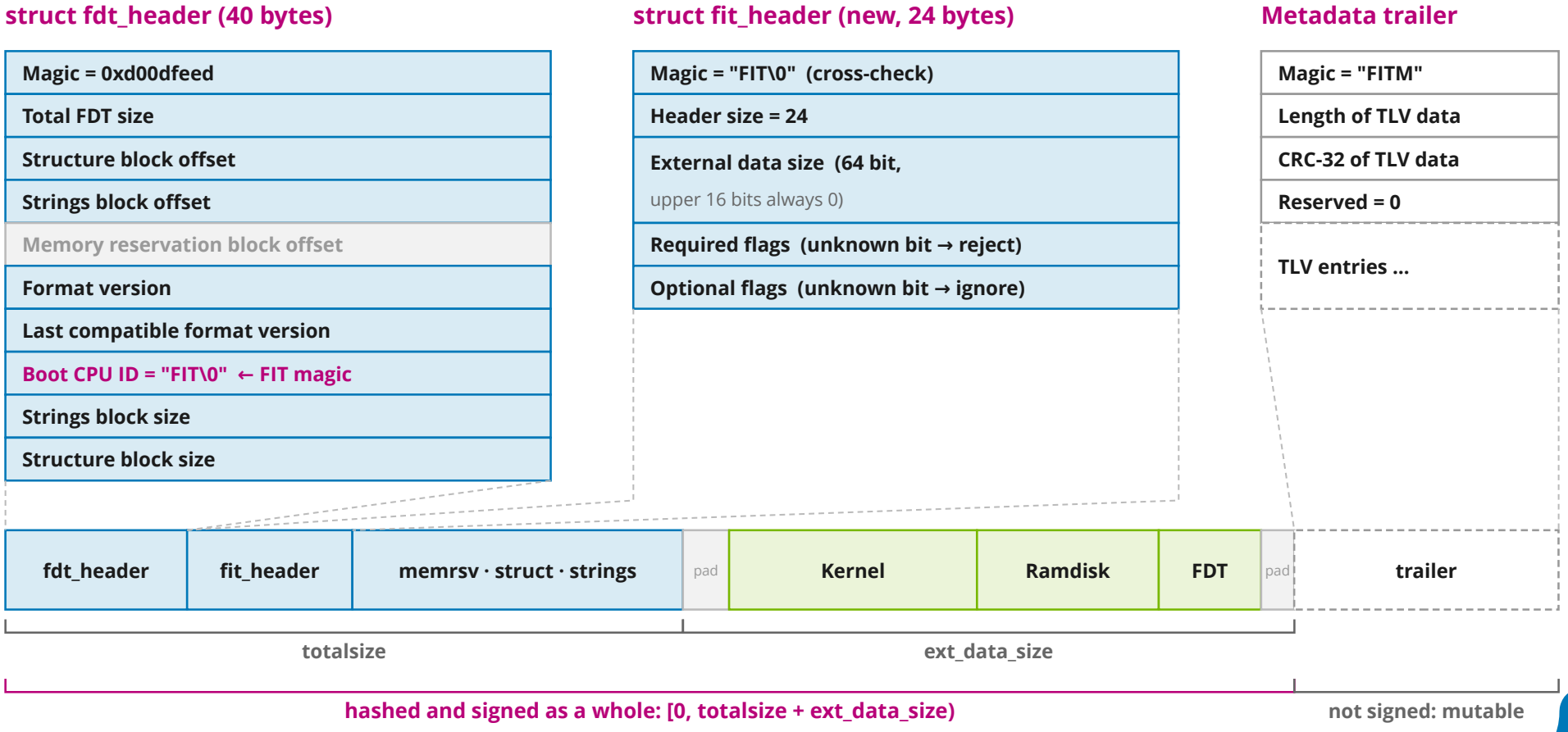
Whole-FIT Signing

- Not a new idea!
  - U-Boot i.MX8M HABv4: sign the whole FIT as firmware
  - Android Verified Boot (U-Boot's implementation is TOCTOU-vulnerable)
  - `CONFIG_IMAGE_PRE_LOAD_SIG` (since v2022.07, barely documented)

- What I think is needed:
  - Something generic, documented, *backwards-compatible*
  - Address the most common use-case: kernel boot container
  - Attractive even without signing:
    - detect a FIT
    - know its size
    - append metadata
  - No FDT parsing before signature verification

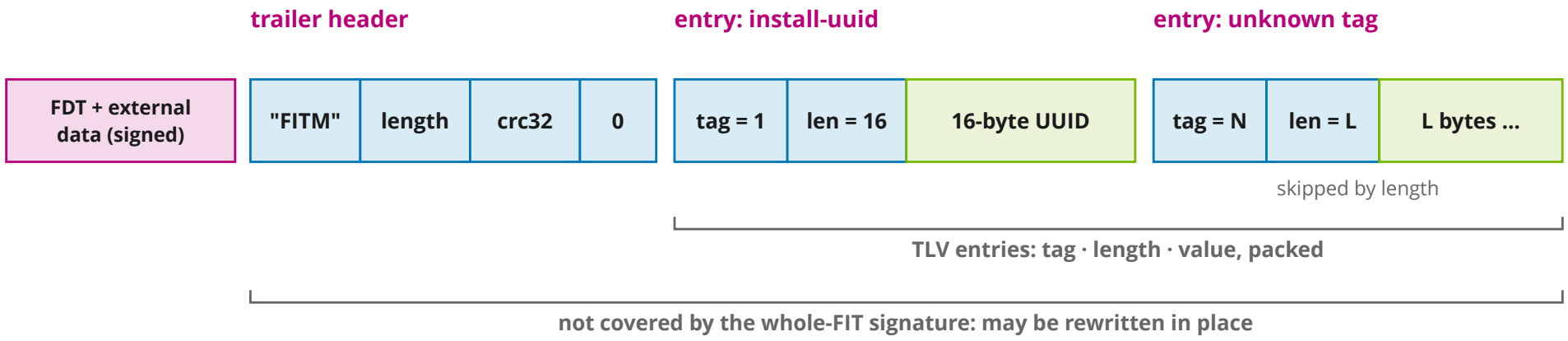


A FIT that says FIT on the tin



FIT metadata trailer: TLV

- Simple tag, length, value format: unknown tags are skipped
- `install-uuid`: mutable metadata without breaking the signature
- Room for detached signatures



Duplicate tags: the last entry wins. Tag 0: no-op (blank an entry in place).

Order of operation

Today: parse, then verify

1) walk FDT, pick a configuration
2) resolve image refs → node list
3) hash those nodes + strings slice
4) verify signature
5) resolve image refs AGAIN, hash, boot

Whole-FIT: verify, then parse

1) read a number of fixed-offset integers, bounds-check
2) hash `[0, totalsize + ext_data_size)`
3) verify signature
   ------- trust boundary -------
4) NOW parse the FDT: it is your own signed data



Compatibility Story

- **New** bootloader + **old** FIT: per-configuration signature verification, for a transitional period
- **Old** bootloader + **new** FIT: ignores the FIT header anyway, per-configuration signature verification
- **New** bootloader + **new** FIT: verifies the whole-FIT signature
- Sign both ways during the transition
- Policy determines what kind of signature is required:
  should never be inferred from the FIT magic



Where are we at?

- Consensus reached with Simon Glass on the approach
  - Competing PR #51 closed in favour of PR #44
- Ball is in my court
- After this lands: Define TLV signature entry, Implementation

Interested in collaborating or sponsoring? Talk to me!



Survival of the FITtest

(Why you should test that FIT fits)



Vulnerabilities will happen

- Bootloader updates need to be easier
  - e.g. with RAUC's power-fail-safe bootloader update methods
  - Process changes desirable on the bootloader side. For barebox:
    - Critical fixes this year's got Security advisories
    - Requesting CVEs for critical issues, e.g. CVE-2026-33243
    - backporting to LTS release: v2026.04.y
    - Make updates easier with Migration guides
- Testing for *pitfalls* needs to be easier
  - Do you test that an unsigned FIT is indeed refused?

Missing config option or mkimage -r flag may make you silently insecure!



LLMs increase the urgency

- LLMs change the equation IMO. Exploitation can go like this:
  - Here is a signed FIT
  - Here's the open source repo and the patch fixing an issue
  - Give me a reproducer that a vulnerable bootloader still accepts
- Widely accessible - Doesn't even need a "Cyber" model.

Here's something I vibecoded back *in March*...



FIT Manipulation as a Service

fitfall.de



Drop a FIT image here to manipulate it
or click to browse

- [x] Change DT `/model`
- [ ] Replace Kernel
- [ ] Append Initramfs

Signature handling: Redirect the signed hashed-nodes list

root!

Download rooted FIT image

https://fitfall.de

Thanks



https://fitfall.de

Questions?