Bootloaders Under Fire

Real-World Threats and Practical Defenses

EMBEDDED
LINUX
CONFERENCE

Ahmad Fatoum – a.fatoum@pengutronix.de

About Me

Ahmad Fatoum

Pengutronix

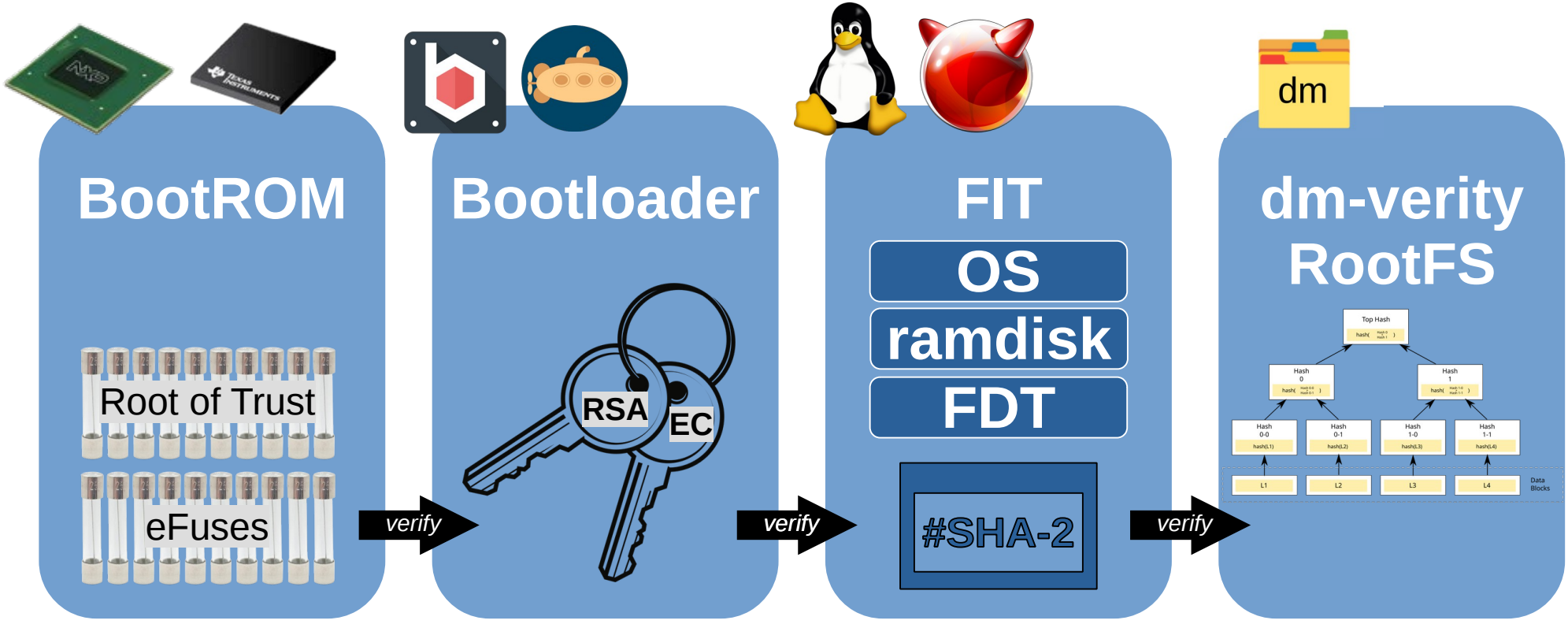
a3f

a.fatoum@pengutronix.de

@a3f@fosstodon.org

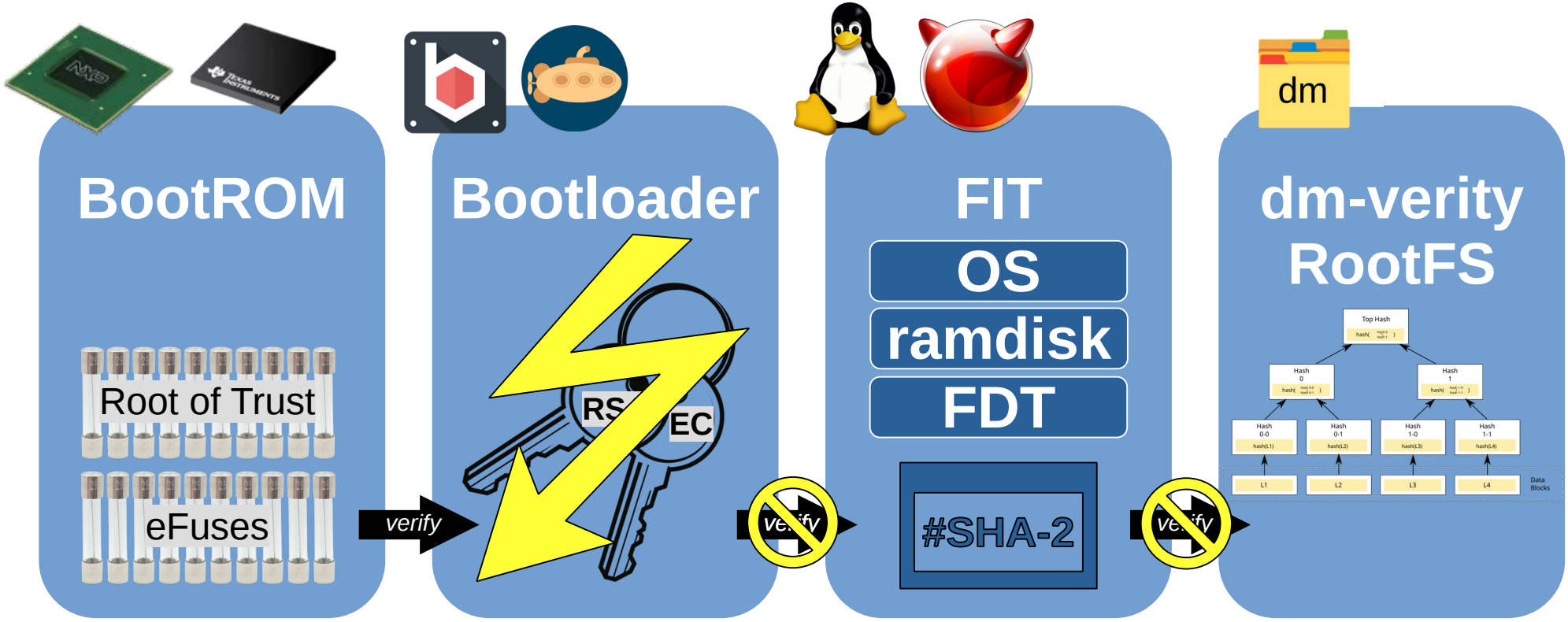
- Kernel and Bootloader Porting
- Driver and Graphics Development
- System Integration
- Embedded Linux Consulting

An Example Verified Boot Chain



https://www.amazon.de/-/en/SIBA-Pack-Glass-Fuses-Characteristics/dp/B0DG6TL249?th=1

A Broken Verified Boot Chain

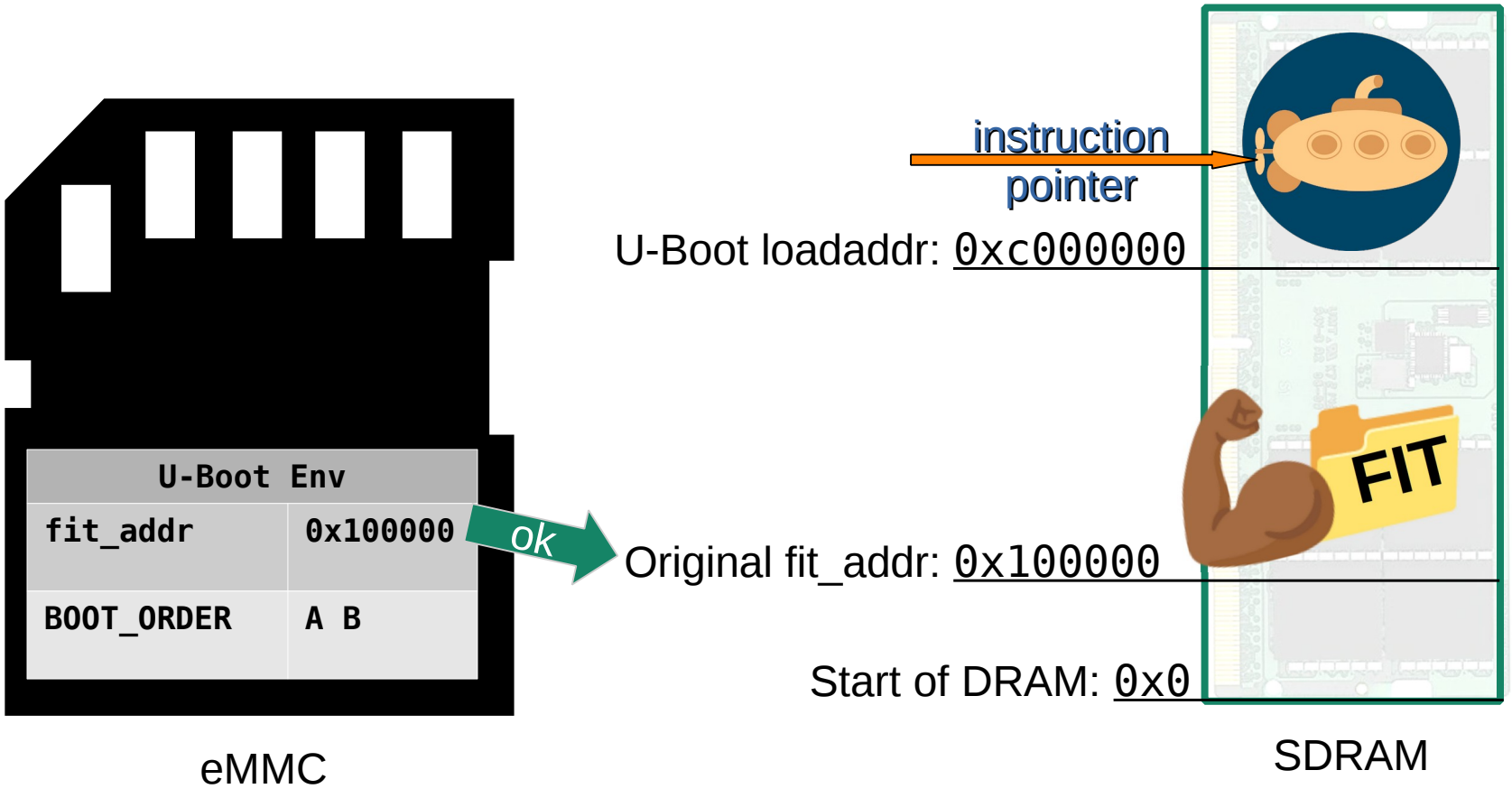


https://www.amazon.de/-/en/SIBA-Pack-Glass-Fuses-Characteristics/dp/B0DG6TL249?th=1

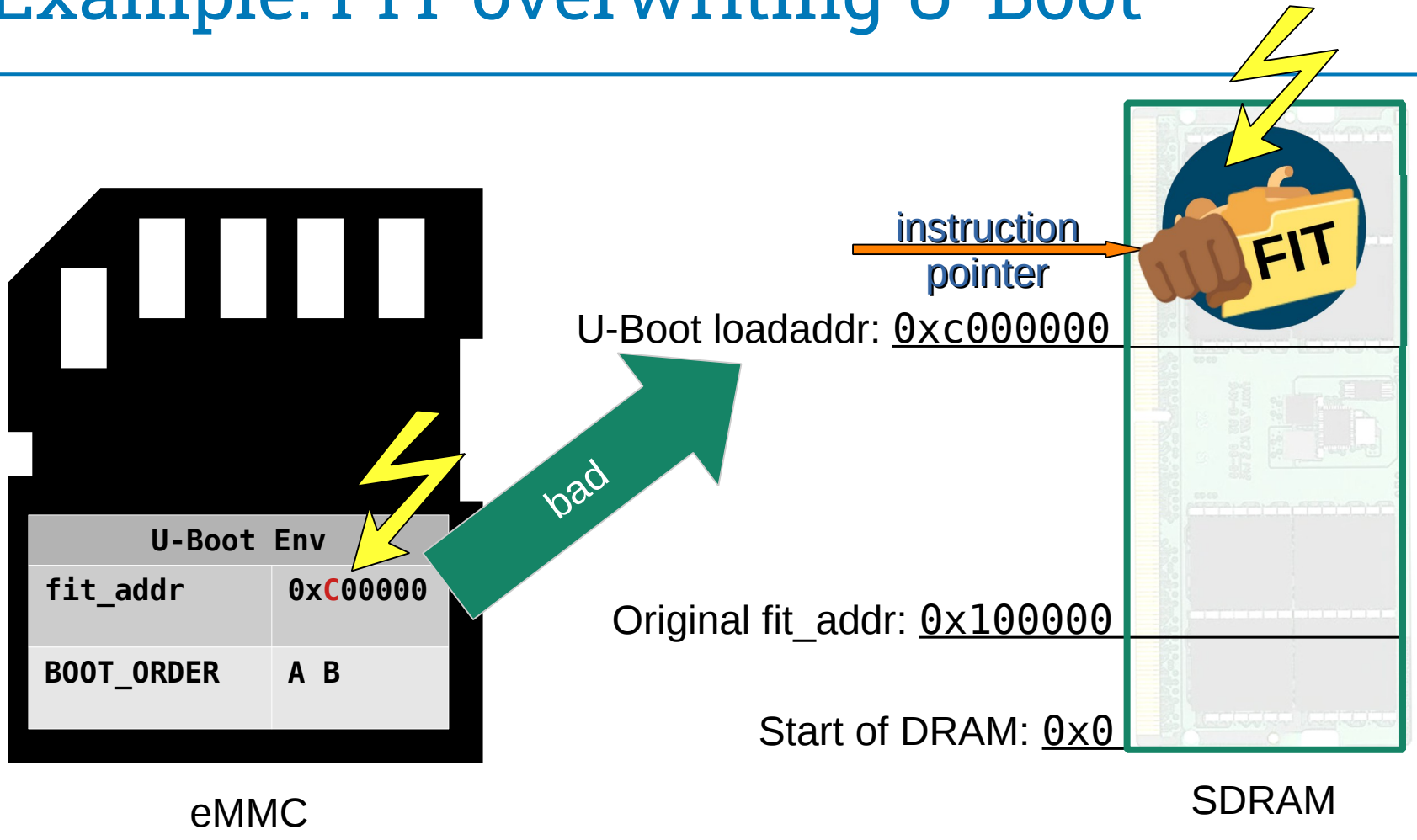
Exploit Example #1

— Hardly Hardened —

Example: FIT overwriting U-Boot



Example: FIT overwriting U-Boot



Takeaways

- A lot went wrong to get us here
  - Environment was not filtered[1]
  - `CONFIG_LMB` not doing what it should/could?
  - Code mapped writable, actually everything was mapped RWX
- But some things went right!
  - Only SPL was running at EL3. Attacker only gained control over EL2 (i.e., keys in OP-TEE not compromised)

[1] and in U-Boot is never authenticated

Restricting Boot State

- Takeaway: Whitelist Environment Variables!
  - In U-Boot via: `env import $addr [var …]`
    - Unfortunately, no global environment whitelist possible
  - Alternative: dedicated boot state as in barebox
    - persistent environment can be disabled completely leaving only built-in environment + barebox-state[1] for boot state

[1] for which barebox supports authentication

MMU Protection?

- ARM bootloaders usually enable the MMU to benefit from caches
  - but didn't traditionally use it for protection..

MMU protection!

- ARM bootloaders usually enable the MMU to benefit from caches
  - but didn't traditionally use it for protection..

- New in ~~Windows 95~~ bootloaders in 25: MMU protection and W^X
  - `CONFIG_MMU_PGPROT` in U-Boot >= v2025.07 for arm64
  - `CONFIG_ARM_MMU_PERMISSIONS` in barebox >= v2025.08 for both arm32/arm64

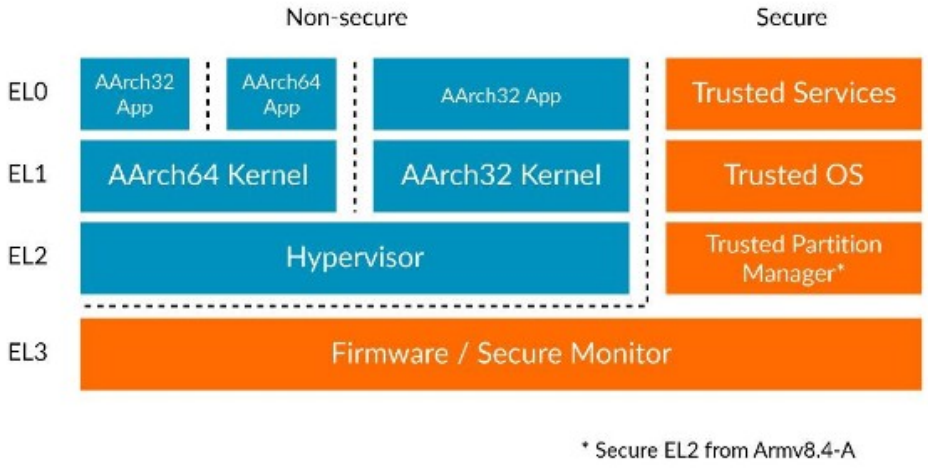
W^X is not enough

W^X is not enough

- Control flow integrity wholly missing
  - return-oriented programmers will still have a field day
  - Case in point: Stack overflow protection:
    - Needs HWRNG drivers: e.g., barebox still has no 64-bit CAAM driver
    - Needs canaries to actually be randomized: U-Boot hardcodes `0xfeedf00ddeadbeef`

Trusted Computing Base (TCB)

- ARMv8-A processors start executing at *EL3*
  - Less code in EL3 = Less attack surface = Tighter TCB = ***Good!***
- On i.MX8M, usually:
  - EL3: U-Boot SPL/barebox PBL, TF-A
  - S-EL1: OP-TEE
  - EL2: Proper bootloader → Linux
- Falcon boot *increases* the TCB!
- Late OP-TEE loading deprecated in barebox



Exploit Example #2

– All buzz, no fuzz –

File System Issues

> Be a bootloader

1) Have File System ported from Linux Kernel

File System Issues

> Be a bootloader

1) Have File System ported from Linux Kernel
2) Watch Linux kernel developers, users, researches collaborate to fix issues

File System Issues

> Be a bootloader

1) Have File System ported from Linux Kernel
2) Watch Linux kernel developers, users, researches collaborate to fix issues
3) Do nothing.

File System Issues

> Be a bootloader

1) Have File System ported from Linux Kernel
2) Watch Linux kernel developers, users, researches collaborate to fix issues
3) Do nothing.
4) Collect ~~Profit~~ CVEs

- CVE-2024-57262
- CVE-2025-26721
- CVE-2025-26722
- CVE-2025-26723
- CVE-2025-26724
- CVE-2025-26725

- CVE-2024-57256
- CVE-2024-57257
- CVE-2024-57259
- CVE-2024-57254
- CVE-2024-57255
- CVE-2024-57258

Takeaways from FS vulnerabilities

- File Systems are easily the most complex parsers in a bootloader
- Linux won't allow you to mount them without root
  - Your bootloader doesn't have the same hardening measures Linux does
- Linux users have a way out with dm-verity and dm-integrity
  - No support in either barebox or U-Boot
- Logical conclusion: Do **NOT** put a signed kernel in a unsigned file system
  - But still, the keeping the code up-to-date story is subpar

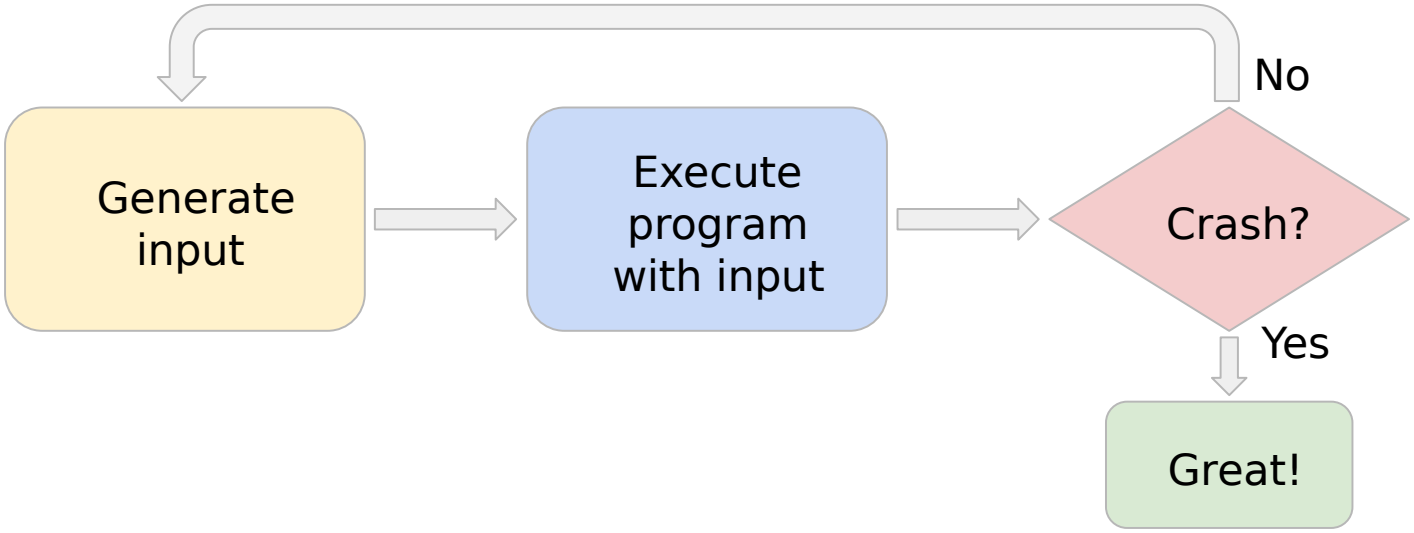
There is no shortage on parsers though..

Untrusted Bootloader Inputs beyond FS

- What code is security sensitive?
  - Factory data in EEPROM
  - Partition tables, Filetype detection (in barebox)
  - Bootloader environment and boot state
  - Flattened Image Tree (FIT)
  - Over-the-wire protocols like USB

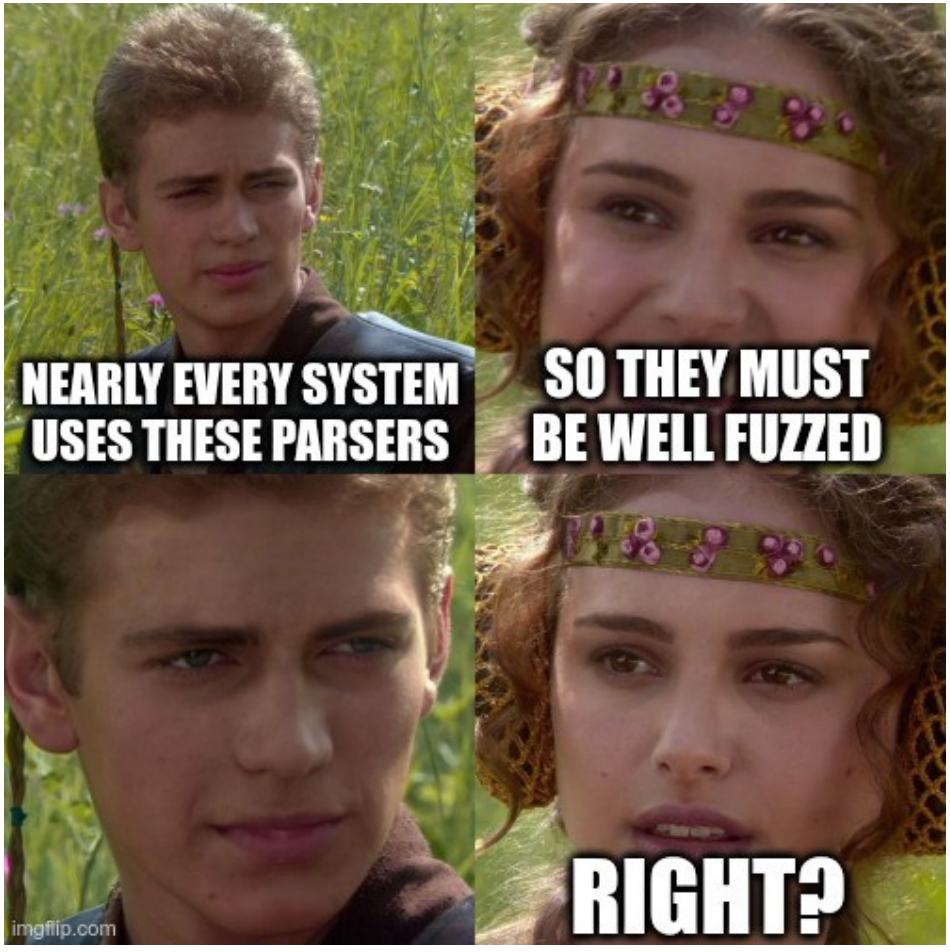
Prime opportunity for fuzzing!

Fuzzing



https://2022.bsidesmunich.org/talks/001_04-V7HFMA-fuzzing_usb_with_raw_gadget/

Fuzzing


NEARLY EVERY SYSTEM
USES THESE PARSERS
SO THEY MUST
BE WELL FUZZED
RIGHT?
imgflip.com

Fuzzing across boot loaders and firmware

- Coreboot: Single fuzz test for JPEG boot splashes
- TF-A: only fuzzes secure monitor calls[1]
- U-Boot: Single fuzz test for Virt I/O rings[2]
- Grub: Patches on ML, never merged
- barebox v2025.07.0: Fuzzing infrastructure and tests for 5 critical parsers upstreamed
  - Found dozens of issues across partition parsing, FIT and DT parsing

---

[1] Nothing for TF-A-as-BL2, e.g., as used for STM32MP1

[2] apparently part of some abandoned Google research

Writing a fuzz test in barebox

```
#include <fuzz.h>

static int fuzz_filetype_detect(const u8 *data,
                                size_t size)
{
        file_detect_type(data, size);
        return 0;
}
fuzz_test("filetype", fuzz_filetype_detect);
// `-- or fuzz_test_ramdisk, fuzz_test_str
```

```
$ LLVM=-19 make libfuzzer_defconfig
$ ./barebox --fuzz "filetype"
```

Exploit Example #3

— Environmentally Conscious —

Environmental Consciousness

- GPT replaces the 8-bit MBR partition type with a Type GUID

```
root@DistroKit:~ lsblk -o NAME,PARTLABEL,PARTTYPE /dev/mmcblk0

NAME         PARTLABEL            PARTTYPE
mmcblk0
|-mmcblk0p1  barebox-environment 6c3737f2-07f8-45d1-ad45-15d260aab24d
|-mmcblk0p2  barebox-state       4778ed65-bf42-45fa-9c5b-287a1dc4aab1
|-mmcblk0p3  root-A              69dad710-2ce4-4e3c-b16c-21a1d49abed3
|-mmcblk0p4  root-B              69dad710-2ce4-4e3c-b16c-21a1d49abed3
`-mmcblk0p5  data                da7af157-d639-49e9-89a0-fc74beeb8954
```

- .. and barebox conveniently started making use of it:

```
Commit 9f868f78bc54
("environment: use barebox environment from GPT partitions")
```

Takeaways from environment autoloading

- Exposure was hopefully limited
  - boot state *should* lives outside environment
  - Secure systems *should* rely only on built-in environment and disable runtime persistence environment (`CONFIG_ENV_HANDLING`)

- Lots of *should*s. How can users find out?

Understanding Risk: Documentation

- What's self-evident to developers isn't necessarily so for users
- I started writing down security considerations for barebox:

  https://www.barebox.org/doc/latest/user/security.html
- and recorded some recommendations in Kconfig

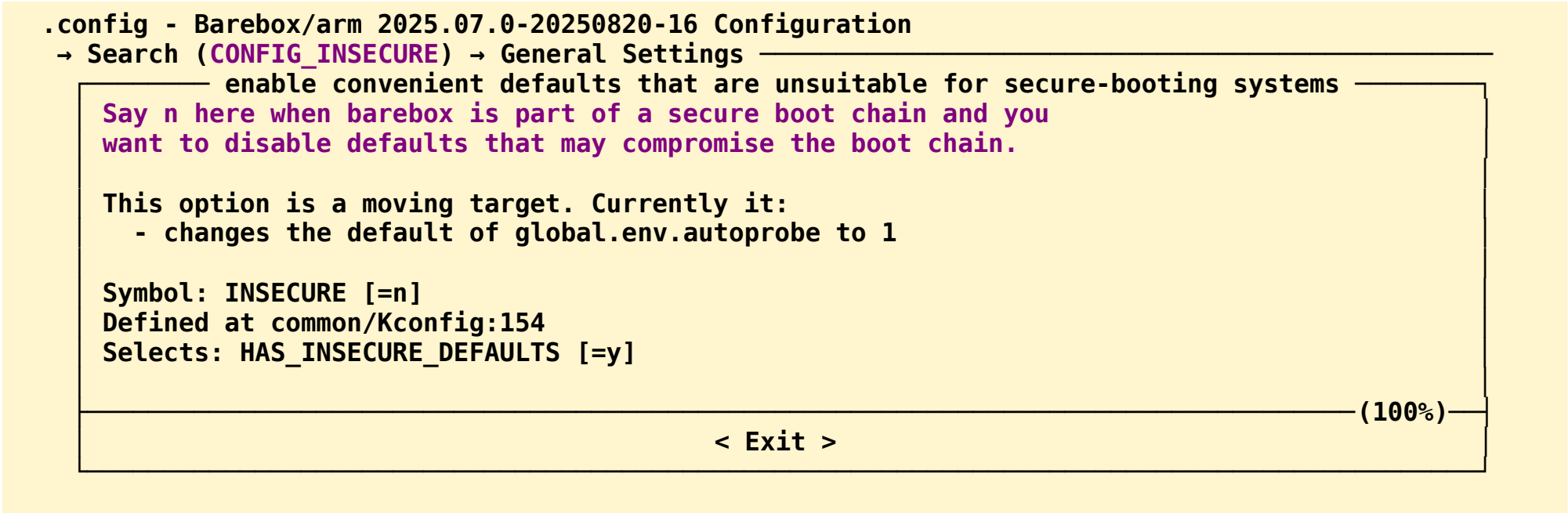
barebox 2025.07.0 documentation

BAREBOX

Table of Contents

1.21. Security Considerations
- 1.21.1. Verified Boot
- 1.21.2. Ensuring barebox is verified
- 1.21.3. Loading firmware
- 1.21.4. Ensuring the kernel is verified
  - 1.21.4.1. Disabling the shell
  - 1.21.4.2. Disabling mutable environment handling
  - 1.21.4.3. Avoiding use of file systems
- 1.21.5. Configuring barebox
  - 1.21.5.1. Compile-time configuration
  - 1.21.5.2. Run-time configuration
- 1.21.6. Security Policy

Security vs Convenience



You better have a second look on these

```
.config - Barebox/arm 2025.07.0-20250820-16 Configuration
 → Search (HAS_INSECURE_DEFAULTS) ──────────────────────────────────────────────
   ┌──────────────────────────────── Search Results ─────────────────────────────┐
   │ Type  : bool                                                               │
   │ Defined at common/Kconfig:144                                              │
   │ Selected by [y]:                                                           │
   │   - BOOTM_OPTIONAL_SIGNED_IMAGES [=y] && BOOTM [=y]                        │
   │   - ENV_HANDLING [=y]                                                      │
   │   - FASTBOOT_CMD_OEM [=y] && FASTBOOT_BASE [=y]                            │
   │ Selected by [n]:                                                           │
   │   - MACH_K3_CORTEX_R5 [=n] && ARCH_K3 [=n] && 32BIT [=y] && !FIRMWARE_VERIFY_NEXT_IMAGE [
   │   - INSECURE [=n]                                                          │
   │   [ etc... ]                                                               │
   │                                                                      (100%)│
   │                                  < Exit >                                  │
   └────────────────────────────────────────────────────────────────────────────┘
```

Understanding Risk

```
.config - Barebox/arm 2025.07.0-20250820-16 Configuration
──────────────────────────── Enable OEM commands ────────────────────────────
CONFIG_FASTBOOT_CMD_OEM:

This option enables the fastboot "oem" group of commands. They allow to
executing arbitrary barebox commands and should be disabled in secure
environments.

Symbol: FASTBOOT_CMD_OEM [=y]
Selects: HAS_INSECURE_DEFAULTS [=y]
──────────────────────────────────────────────────────────────────────(100%)──
                                  < Exit >
```

Exploit Example #4

— Wasted by E-Waste —

Factory Images

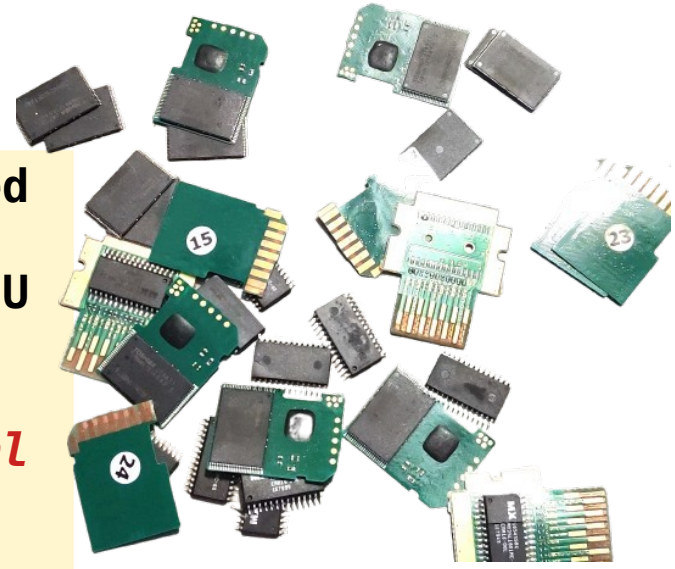
- Setup in factory often needs to run with secure boot chain active
  - *e.g., Hardware Unique Key (HUK)* in TrustZone only accessible after secure boot is enabled

Nintendo: *Let's create signed factory images and dispose of them afterwards*

Paid the Beak

DeadlyFoez writes[1] on 17.06.2025:

> **Nintendo tried to destroy [the SD Cards used in the Nintendo factory setup process for installing the software to the Wii and Wii U systems] by crushing them and bending them in the middle.**
> ***About 25% of the cards were still functional*** **with a little straightening and convincing and I was able to recover the data.**



[1] https://consolebytes.com/wii-u-sdboot1-exploit-paid-the-beak/

Takeaways: changing security expectations

- Security expectations are not set in stone
  - **Development**: Need to allow debugging access, but not differ too much from in-field use
  - **Factory**: May need to allow access to the system, e.g., to export public keys
  - **In-the-Field**: Locked down, i.e., disallow unverified software from running or detect it
  - **Field-Return**: (Permanently) unlock single device: Debugging, GPLv3 compliance, right to repair
  - **Trade Fair**: Take off-the-shelf device from distributor, reflash and reset to field mode after fair

Takeaways: changing security expectations

- Different binaries for different use cases
  - + Non-existent code is unexploitable!
  - – Multiple image increases complexity for testing and maintenance
- Same binary with runtime decisions
  - + Easier handling across development, Testing and in the factory
  - – Runtime disablement of features not as straight-forward-as-Kconfig

Introducing Security Policies

- Add empty policies to the build system
- Run `make security_menuconfig` to fill them out one-by-one

```
security/myboard-lockdown.sconfig.tmp - Barebox Security Configuration
→ General Settings ──────────────────────────────────────────────────────────────
                                  General Settings
  Arrow keys navigate the menu.  <Enter> selects submenus ---> (or empty submenus
  ----).  Highlighted letters are hotkeys.  Pressing <Y> includes, <N> excludes, <M>
  modularizes features.  Press <Esc><Esc> to exit, <?> for Help, </> for Search.
  Legend: [*] built-in  [ ] excluded  <M> module  < > module capable
  ┌──────────────────────────────────────────────────────────────────────────────┐
  │       [ ] Allow console input                                                │
  │       [*] Allow executing shell scripts                                      │
  │       [ ] Allow loading barebox environment from persistent media            │
  │       [ ] Allow Fastboot OEM commands                                        │
  └──────────────────────────────────────────────────────────────────────────────┘
  ──────────────────────────────────────────────────────────────────────────────
              <Select>    < Exit >    < Help >    < Save >    < Load >
```

Introducing Security Policies

- The result is Kconfig style files you can commit

```
SCONFIG_POLICY_NAME="lockdown"
SCONFIG_SECURITY_POLICY_SELECT=y

# General Settings
# SCONFIG_CONSOLE_INPUT is not set
SCONFIG_SHELL=y
# SCONFIG_ENVIRONMENT_LOAD is not set
# end of General Settings

# Boot Policy
# SCONFIG_BOOT_UNSIGNED_IMAGES is not set
# end of Boot Policy
```

Introducing Security Policies

- Board code decides policy as it sees fit: eFuses[1], JWT[2], Secure World[3] ....etc.

```
/* Check if secure boot enabled */
if (imx_hab_device_locked_down())
     security_policy_select("lockdown");
else
     security_policy_select("factory");
```

---

[1] e.g., a factory-done fuse or the number of non-revoked keys
[2] e.g., Android Verified Boot TA in v2025.05.0 onwards
[3] added with v2023.12.0

Introducing Security Policies

- Generic code consults policies as needed:

```
int getchar(void)
{
  if (!IS_ALLOWED(SCONFIG_CONSOLE_INPUT))
    return -1; /* or -EPERM */

  /* do stuff */
}
```

- At v2: https://lore.kernel.org/barebox/20250820-security-policies-v1-0-76fde70fdbd8@pengutronix.de/T/#t

Security Policies

- Collaborate on the upstream mechanisms,
  but customize the concrete policies for *your* project
- Comfortably auditable textual security configuration
- Completeness enforced: After updates, you must decide on values for new options, otherwise:

```
*** Security policy myboard-lockdown.sconfig was not up to date.
***
*** Please run "make security_olddefconfig" or
*** another configurator and commit the results to
*** security/myboard-lockdown.sconfig
make: *** [Makefile:185: sub-make] Error 2
```

Demo Time?

- Come see it in action at the Pengutronix showcase today

- Featuring colored USB sticks with JSON web tokens and funky LEDs

Curious for more?

The Bootloader: An Underestimated Risk To Embedded Linux Security - Richard Weinberger, sigma star gmbh

Wednesday August 27, 2025 14:10 - 14:50 CEST

G104 (Level 1)

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?