Secure by Default?

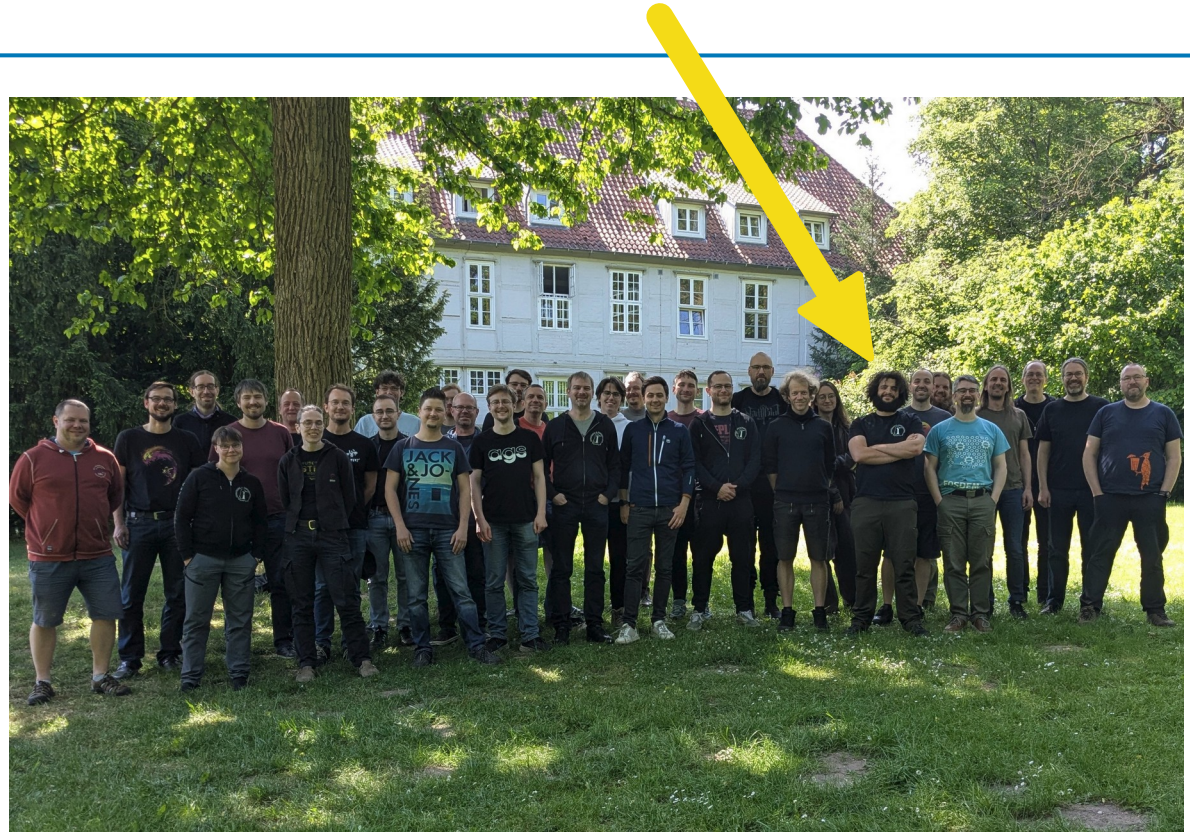
Unified Bootflows from Factory to Field

Ahmad Fatoum – a.fatoum@pengutronix.de

About Me

- Ahmad Fatoum
- Pengutronix
- a3f
- a.fatoum@pengutronix.de
- @a3f@fosstodon.org

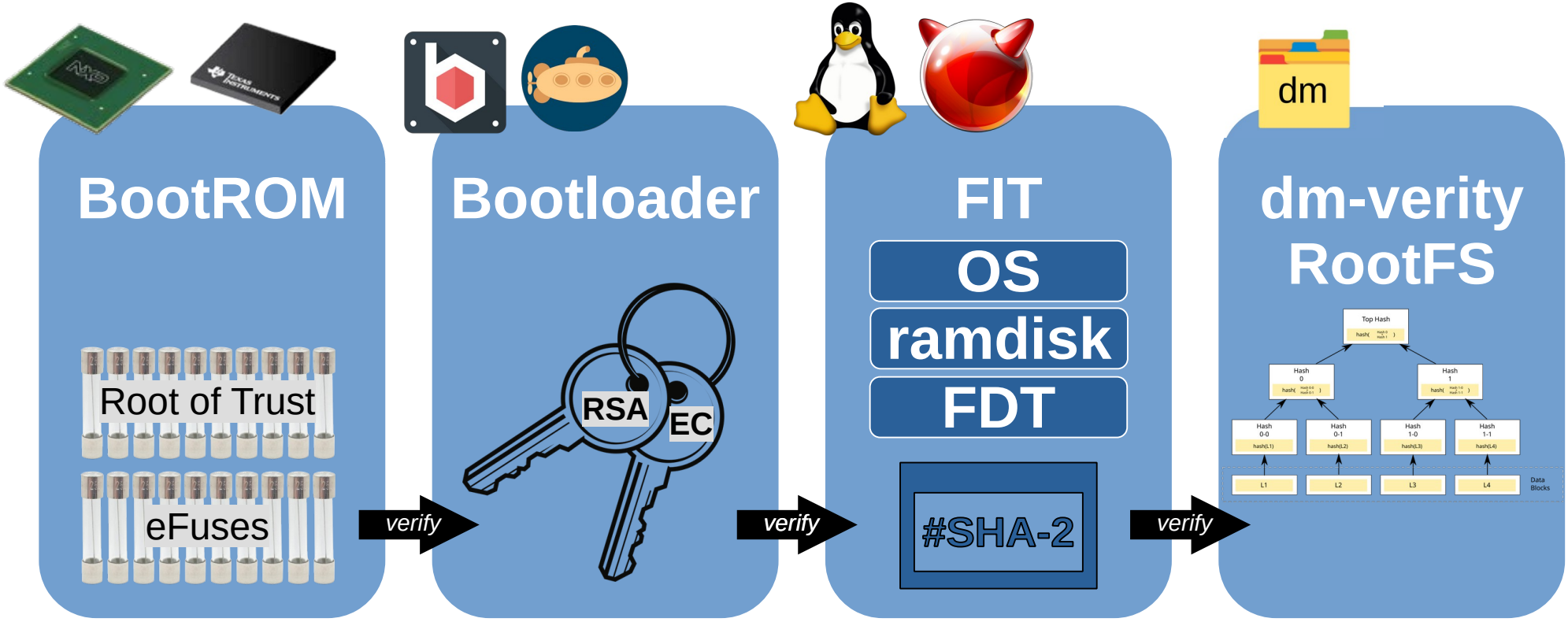
- Kernel & Bootloader Porting
- Driver and Graphics Development
- System Integration
- Embedded Linux Consulting



Verified Boot 📈

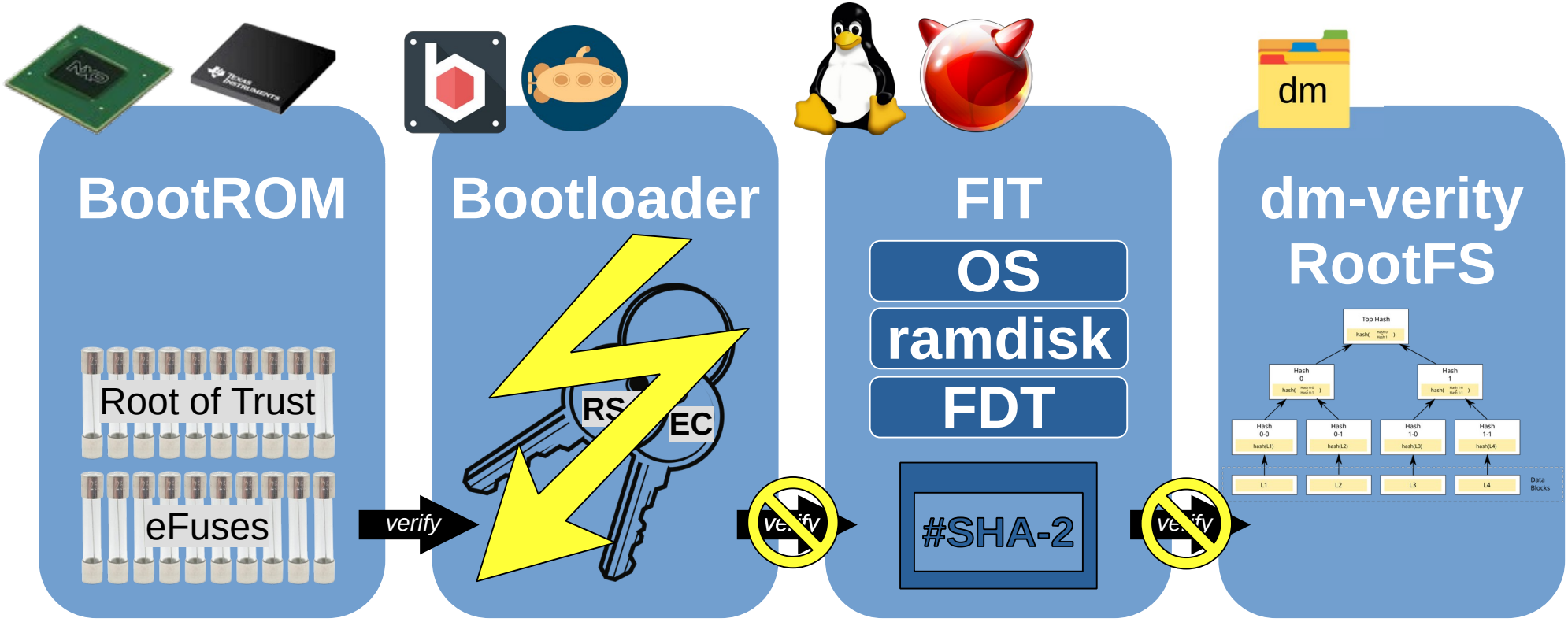
- Most new embedded projects use a secured the boot chain
  - Often motivated by upcoming EU regulation
- For many embedded systems, this takes the form of a verified boot chain with each component verifying the next

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

What do I mean by "Secure by Default"?

- Avoid privileged image variants

  Securely handle different stages of the life cycle with the same bootloader image

- Limit impact when under attack

  Reduce likelihood and impact of vulnerabilities through hardening and proactive testing

- Rehearse recovery after compromise

  Safely return to a trusted state through well-defined maintenance workflows

No Privileged Variants

Device Life cycle

- Security *must* account for life cycle state

  Development ≠ Provisioning ≠ In-Field ≠ RMA

- Encoding life cycle state solely into the software image is a disaster waiting to happen if the image leaks

Device Life cycle

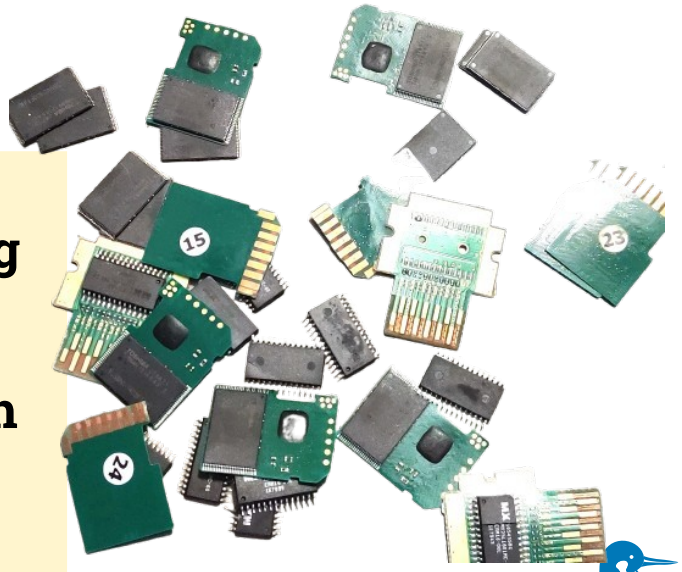
- Security *must* account for life cycle state

  Development ≠ Provisioning ≠ In-Field ≠ RMA

- Encoding life cycle state solely into the software image is a disaster waiting to happen if the image leaks

DeadlyFoez writes1 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.



Device Life cycle

- Security *must* account for life cycle state

  Development ≠ Provisioning ≠ In-Field ≠ RMA

- Encoding life cycle state solely into the software image is a disaster waiting to happen if the image leaks
- Image *should* verify that the state is correct

Device Life cycle

- Security *must* account for life cycle state

  Development ≠ Provisioning ≠ In-Field ≠ RMA
- Encoding life cycle state solely into the software image is a disaster waiting to happen if the image leaks
- Image should ***verify*** that the state is correct
- If we go the extra mile and implement sensible fallback behavior, we can address different situations with the same image

Device Life cycle

- Security *must* account for life cycle state

  Development ≠ Provisioning ≠ In-Field ≠ RMA

- Encoding life cycle state solely into the image is a disaster waiting to happen if the image leaks
- Image should *verify* that the state is correct
- If we go the extra mile and implement sensible fallback behavior, we can address different situations with the same image

- Complexity is only shifted: With life cycle handling in common code, it's more feasible to test it
- In summary, fewer images is better, provided robust ***access control***

Access Control? in the bootloader?!

- Bootloader runtime access control is a mess
- Core issue: Individual threat model defines *'secure'*
- Applying a security policy goes *very* deep into bootloader guts

```
if (lockdown) {
   bootm_force_signed_images();
} else {
   struct console_device *console;
   console = of_console_by_stdout_path();
   console_set_active(console, CONSOLE_STDIOE);
   of_pinctrl_select_state(console->dev->of_node,
   "open");
}
```

- This kind of code is usually not upstreamed

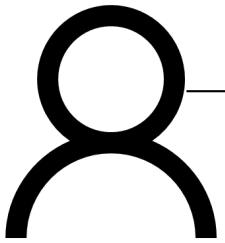
Introducing barebox Security Policies

- Generic code consults the active policy as needed:

```
int getchar(void)
{
   if (!IS_ALLOWED(SCONFIG_CONSOLE_INPUT))
      return -EPERM;

   /* do stuff */
}
```

- Check directly at the security-sensitive operation
  → More future-proof

barebox Security Policies: Visualized



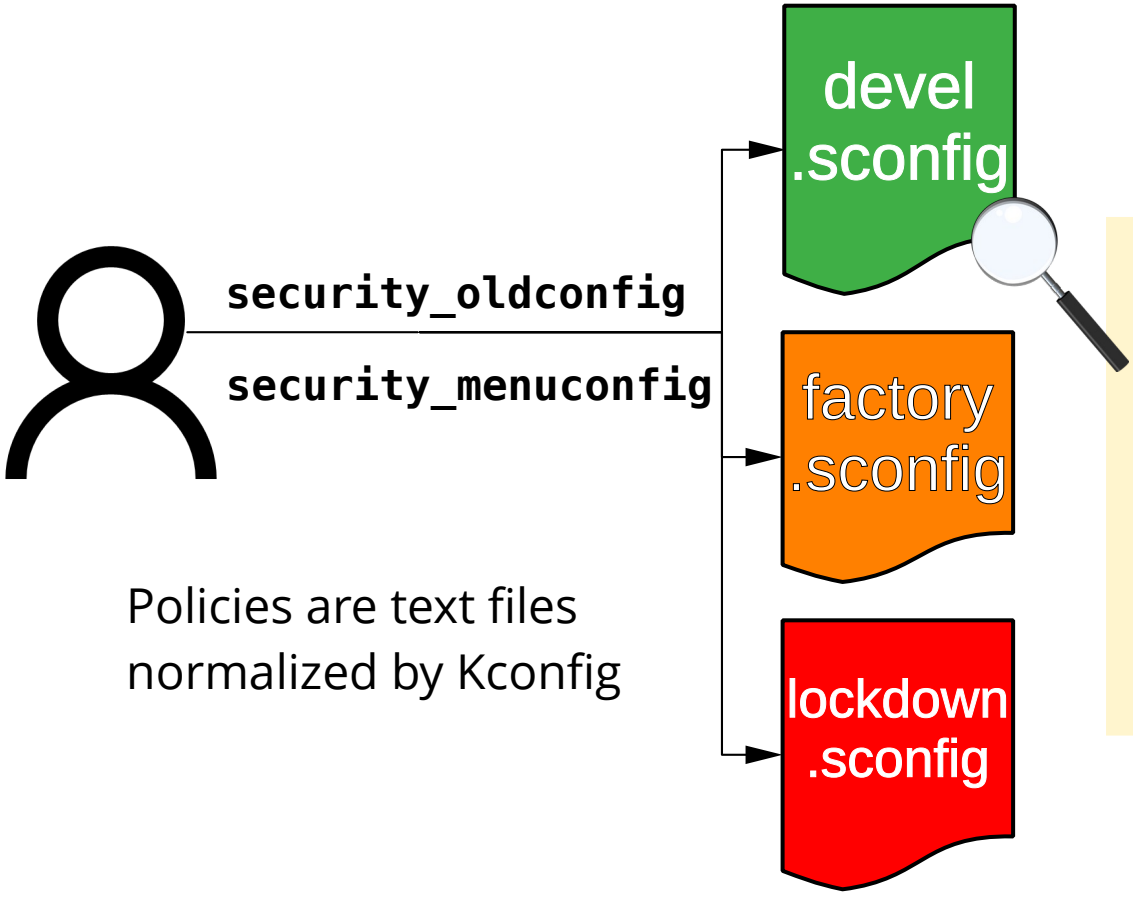
security_oldconfig

security_menuconfig

```
myboard-lockdown.sconfig - Barebox Security Configuration
→ General Settings
                        General Settings
   [ ] Allow console input
   [*] Allow executing shell scripts
   [ ] Allow loading barebox environment from persistent media
   [ ] Allow Fastboot OEM commands

   <Select>    < Exit >    < Help >    < Save >    < Load >
```

make target interactively prompts for one or more security policies

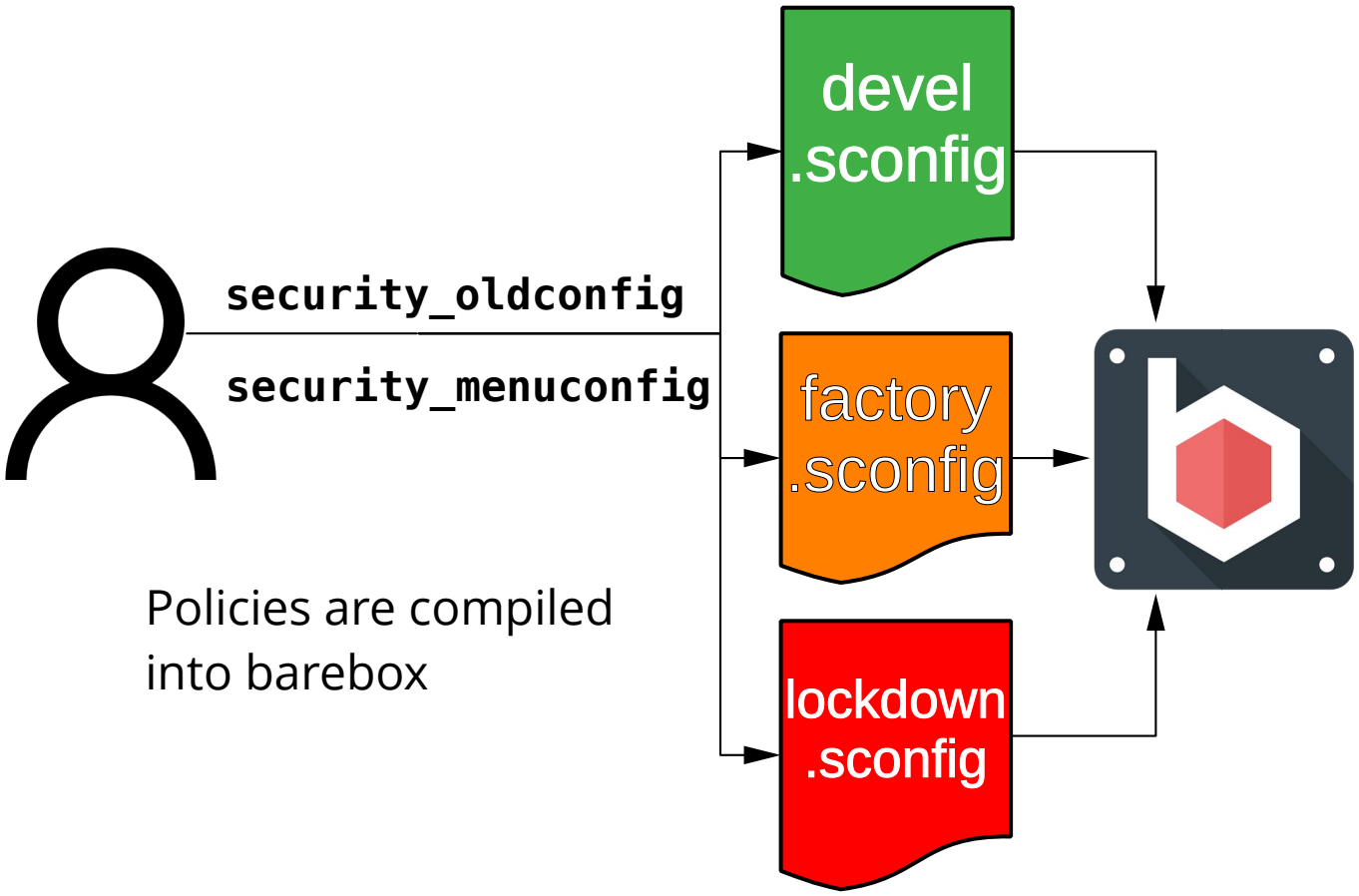
barebox Security Policies: Visualized



Policies are text files
normalized by Kconfig

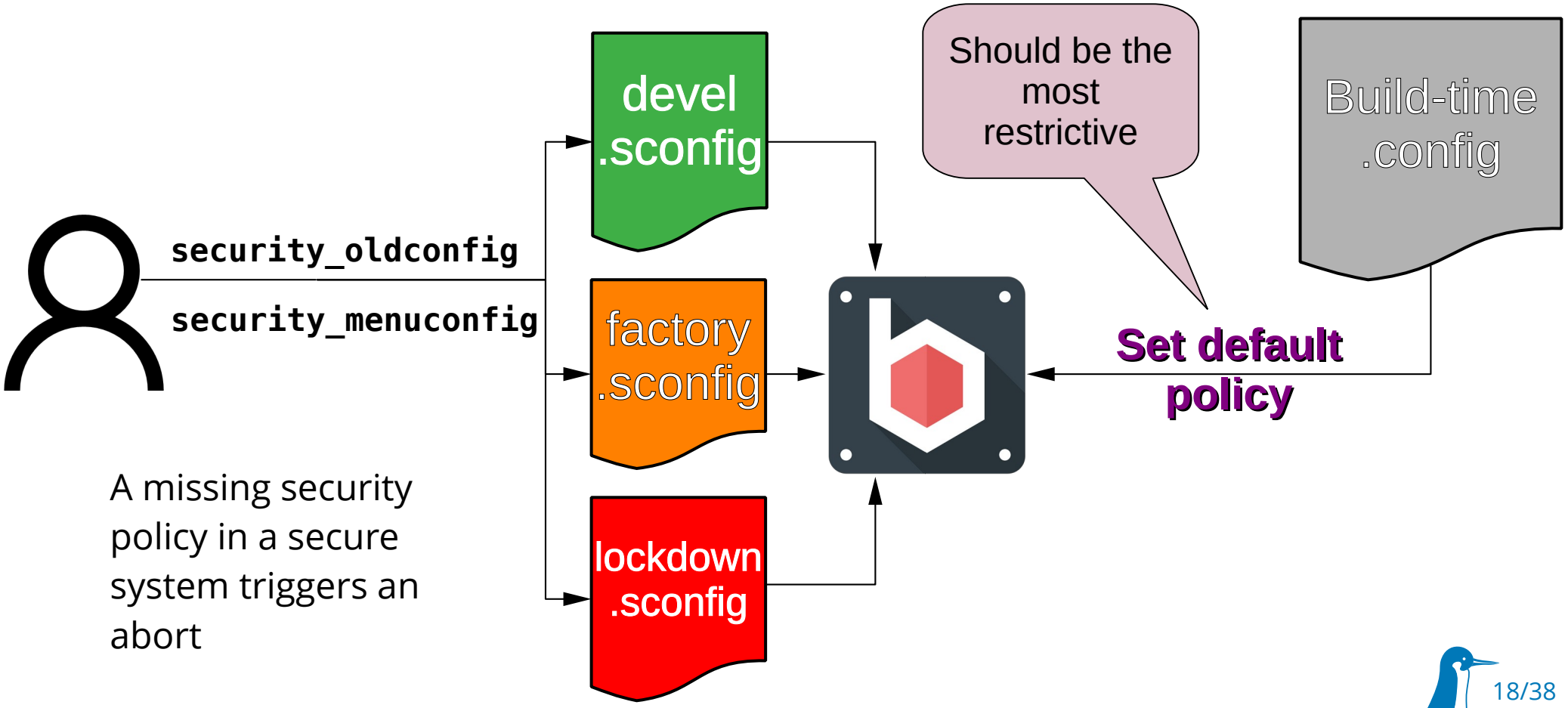
```
SCONFIG_POLICY_NAME="lockdown"
SCONFIG_SECURITY_POLICY_SELECT=y

# 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
```

barebox Security Policies: Visualized



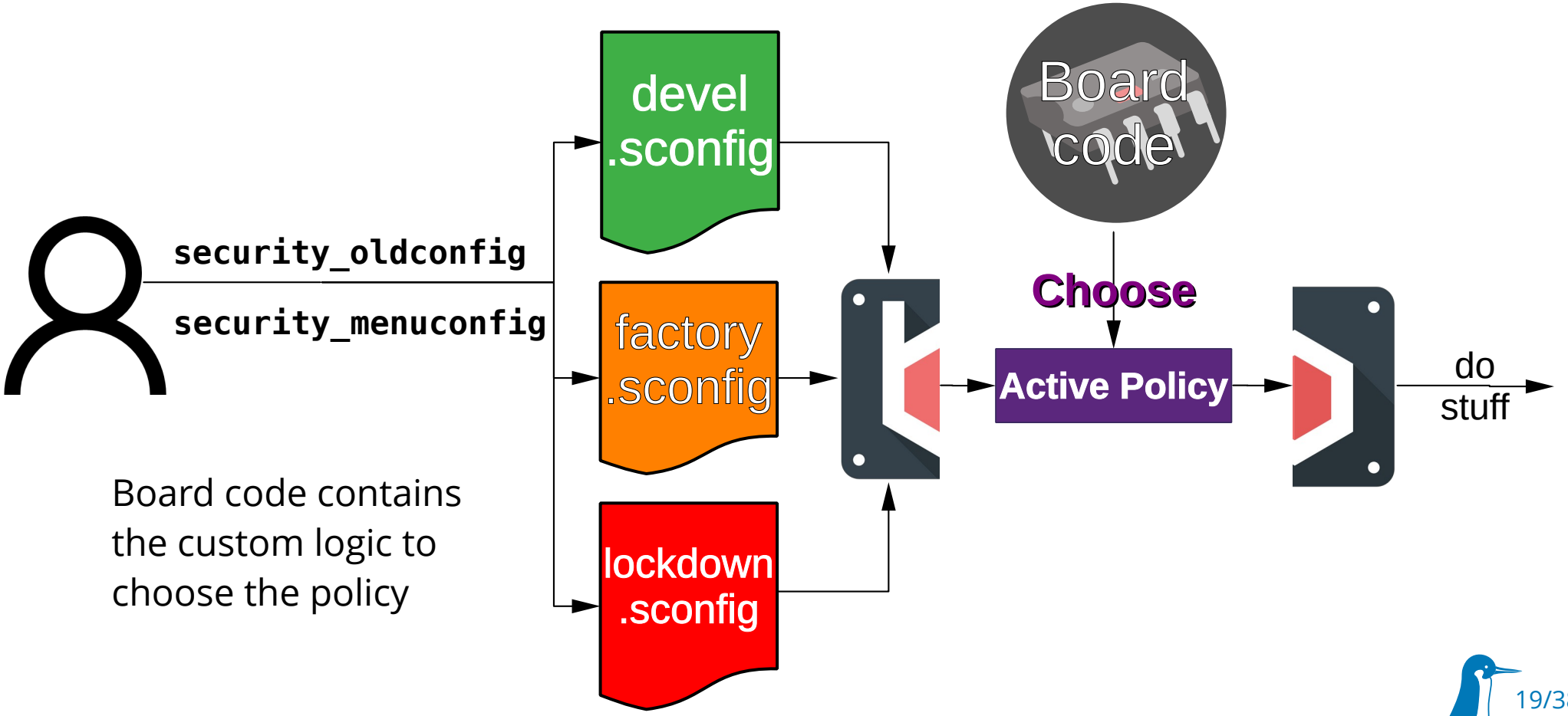
Policies are compiled into barebox

barebox Security Policies: Visualized



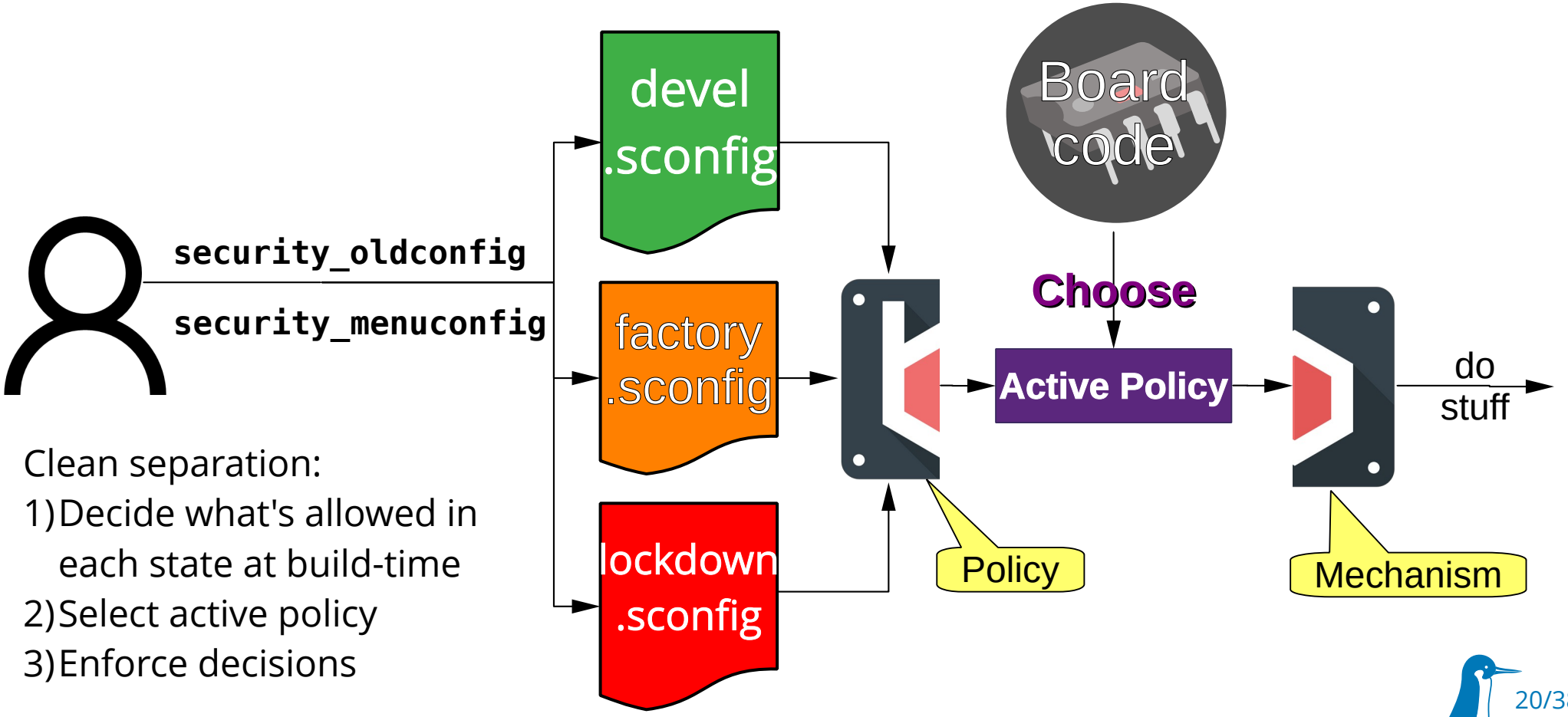
A missing security policy in a secure system triggers an abort

barebox Security Policies: Visualized



Board code contains the custom logic to choose the policy

barebox Security Policies: Visualized



Clean separation:

1) Decide what's allowed in each state at build-time
2) Select active policy
3) Enforce decisions

Fuse-based policy selection

- Board code selects security policy as it sees fit

```
/*
 * 00: Factory mode, straight to devel mode
 * x1: Factory done, no escape
 * 10: Test mode, go to factory done, allow to escape to devel
 */
otp = nvmem_cell_read(factory_nvmem_cell, &len);
if (IS_ERR(otp)) {
        pr_err("Failed to read factory mode: %pe\n", otp);
        return security_policy_select("lockdown");
}

/* no fuse burnt, go to devel mode */
if (!(otp[0] & FUSE_FACTORY_DONE))
        return security_policy_select("devel");

/* Default is lockdown */
```

Unlocking developer devices

- Can also factor in more information than just fuses

```
/*
 * At this point we know that we are in factory done test mode.
 * Ask the user if they want to escape to devel mode.
 */
pr_info("Factory fuse intact. Press <d> to enter devel mode\n");

start = get_time_ns();

while (!is_timeout(start, 5 * SECOND)) {
        if (!console->tstc(console))
                continue;

        c = console->getc(console);
        if (c == 'd') {
                pr_notice("<d> pressed, entering devel mode\n");
                return security_policy_select("devel");
        }
}
```

Unlocking production devices

- Unlock token must be signed
  - So far: JSON Web Tokens (JWT) with RSA signatures
  - New: TLV format and ECDSA signatures 🔗
- Unlock token must not be transferable across devices
  - A SoC unique ID: `barebox_get_soc_uid_bin()` 🔗

- Board code can then find and verify unlock token on boot, from
  - USB flash drive
  - On-disk in an EEPROM (installed beforehand OTA)
  - OP-TEE, e.g. Android Verified Boot TA
  - … etc.

Impact-limiting under attack

Example Heap overflow

```
commit 51eea5c16fd2a829064e852e6ddd4bf73b25b3ac
Author:     Sascha Hauer <s.hauer@pengutronix.de>
AuthorDate: Wed Feb 19 15:18:42 2025 +0100

fs: ext4: fix malloc(size + constant) buffer overflow issues

diff --git a/fs/ext4/ext_barebox.c b/fs/ext4/ext_barebox.c
index 163c4d2fe1b7..7480c045d77e 100644
--- a/fs/ext4/ext_barebox.c
+++ b/fs/ext4/ext_barebox.c
@@ -189,7 +189,7 @@ static const char *ext_get_link(struct
dentry *dentry, struct inode *inode)

        BUG_ON(inode->i_link);

-       inode->i_link = zalloc(inode->i_size + 1);
+       inode->i_link = zalloc(size_add(inode->i_size, 1));
```

Issue has been fixed, but this class of issues is unfortunately quite common in C code

Hardening Measures

- Hardware-assisted Hardening

  e.g. read-only code segments

  - `CONFIG_MMU_PGPROT` in U-Boot >= v2025.07 for arm64
  - `CONFIG_ARM_MMU_PERMISSIONS` in barebox >= v2025.08 for both arm32/arm64

  Also Stack Protector, Guard pages ...etc.
- Still a lot to do bootloader-side, but enable what's already there!
- Proactively eliminate issues
  - Document security considerations to help assess risk
  - e.g. barebox now has upstream fuzz tests for security sensitive parsers like FIT

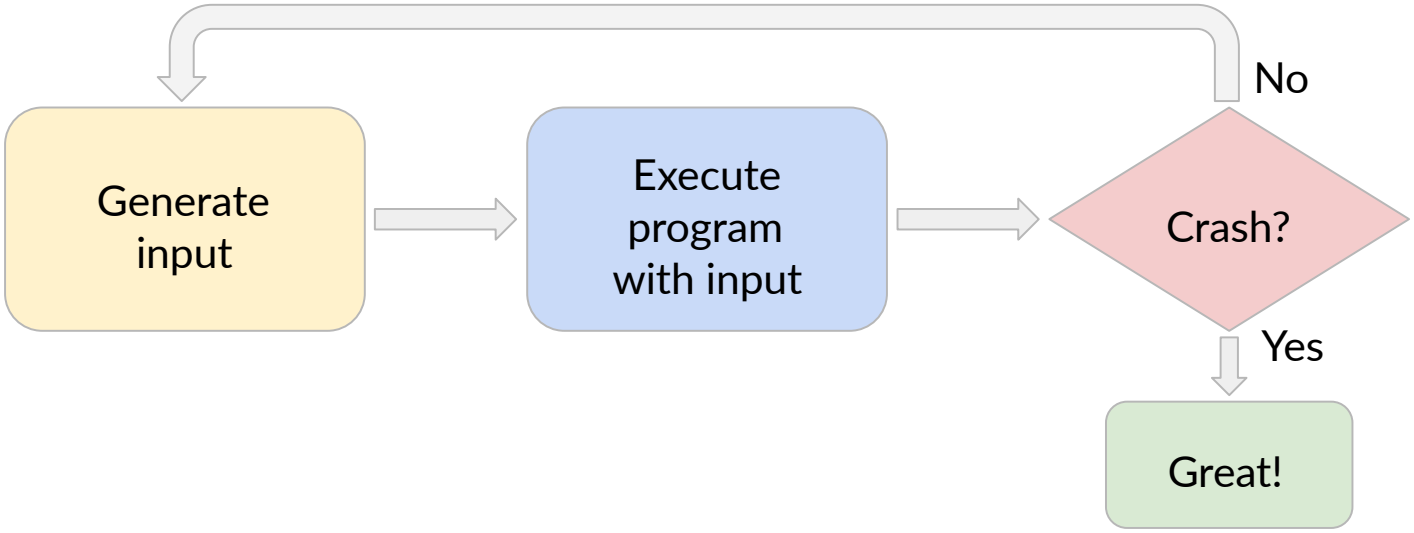
Understanding Risk: Documentation

- What's self-evident to developers isn't necessarily so for users
- Kconfig and (Security Policy) help text are useful, but don't give the whole picture
- We are documenting barebox security considerations:
  https://www.barebox.org/doc/latest/user/security.html

barebox 2025.07.0 documentation

Table of Contents

Fuzzing



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

Fuzz your own security-critical parsers

```
#include <fuzz.h>

static int fuzz_myboard_eeprom(const u8 *data,
                               size_t size)
{
        struct eeprom_data *content;
        content = parse_eeprom_content(data, size);
        free(content);
        return 0;
}
fuzz_test("myboard-eeprom", fuzz_myboard_eeprom);
```

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

Recoverable after Compromise

Updating in the Field

- There will always be (security) bugs
  - The *mechanisms themselves* may be vulnerable
- Projects must be prepared to update vulnerable components at every stage of the boot chain

- Case in point: CVE-2026-33243 🔗
  Critical FIT Signature Verification Bypass Vulnerability
  - Freshly fixed in **barebox v2026.03.1** and **U-Boot v2026.04**
  - Most signed FIT users affected, going back 10+ years

Requirements for the attack

An **attacker with either physical or local access** allowing them to read/write the FIT on the storage medium can **replace any image** in the selected configuration **without having to create a new signature**. The attacker's payload will run at the privilege level of the original payload.

Systems that do not secure the boot chain or do so without using FIT are not affected. For example, Linux v6.10+ can generate FIT images, but as they are not signed, they are unaffected by this vulnerability.

Takeaways

- Critical issues will happen, so you need to be prepared
- Non-technical measures on the project level can go a long way
  - **Documentation** to help assess security impact
  - **Long Term Stable** releases make security updates more predictable
  - **Migration Guides** to reduce risk on updates from one LTS to next
  - **Security Advisories** to help users make informed decisions

barebox LTS

From: Yoann Congal <yoann.congal@smile.fr>
To: openembedded-core@lists.openembedded.org
Subject: [OE-core][whinlatter v2 15/51] barebox/barebox-tools: upgrade 2025.09.0 -> 2025.09.3
Date: Fri, 17 Apr 2026 00:29:58 +0200 [thread overview]
Message-ID: <c113fff232b77e2fd03762e0c694e876933a101b.1776377993.git.yoann.congal@smile.fr> (raw)
In-Reply-To: <cover.1776377993.git.yoann.congal@smile.fr>

From: Ankur Tyagi <ankur.tyagi85@gmail.com>

2025.09.3

Fixed FIT image vulnerability
https://lore.barebox.org/barebox/abljJRMecNdejSD0@pengutronix.de/

Changelog:
https://github.com/barebox/barebox/compare/v2025.09.2...v2025.09.3

2025.09.2

Changelog:
https://github.com/barebox/barebox/compare/v2025.09.1...v2025.09.2

2025.09.1

This stable release is specifically targeted at whinlatter which is
currently at v2025.09.0
https://lore.barebox.org/barebox/aUkaSKDePHF8__LB@pengutronix.de/

Changelog:
https://github.com/barebox/barebox/compare/v2025.09.0...v2025.09.1

Migration Guide

8.20. Release v2026.04.0

8.20.1. Boards

8.20.1.1. TQMA8MPxL

The config option has been renamed from `CONFIG_MACH_TQ_MBA8MPXL` to `CONFIG_MACH_TQ_MBA8MPXX` to accommodate the support for TQMA8MPxS boards with the same binary. The binary has been renamed from `barebox-tqma8mpxl.img` to `barebox-tqma8mpxx.img`.

8.20.1.2. ZynqMP

The Linux v7.0 device trees imported into this barebox release no longer feature an unconditional *"linaro,optee-tz"* compatible OF node.

If OP-TEE use in barebox is desired, this node must be added back to the barebox device tree.

8.20.1.3. HABv4-enablement

Security Advisories

- Common Procedure
  - Security researchers request CVEs and report issue
  - Project provides assessment
  - Issue is fixed, CVE is disclosed
- Problems:
  - Researchers do not necessarily have full overview of implications
  - Not all issues have CVEs assigned
- CVE-2026-33243
  - First CVE requested by the barebox project for itself
  - First advisory to inform users of implications
    - Also covers U-Boot which was affected by the same issue

Further Watching

- Authenticated and Encrypted Storage on Embedded Linux - Jan Lübbe, ELC-E 2019
- Bootloaders under Fire: Real-World Threats and Practical Defenses - Ahmad Fatoum, ELC-E 2025
- Netboot without throwing a FIT - Ahmad Fatoum, FOSDEM 2026

Future Outlook

- Generic "System Data" OP-TEE TA
  - Key/Value-Store with rollback protection / write once
  - Securely configure a different security policy with rollback protection
  - Generic Backend for Bootloader Downgrade Protection
- Reduce FIT-susceptibility to security issues with Whole-FIT signing 🔗
- Work out best practices for Linux userspace use of the new `barebox.security.policy="`*`policy`*`"` kernel parameter

On the web: barebox.org/demo
ML: barebox@lists.infradead.org
Archive: lore.kernel.org/barebox
Github: github.com/barebox
Mastodon: @barebox@fosstodon.org
Matrix: #barebox:matrix.org

barebox security documentation: