FOSDEM'26

Embedded, Mobile and Automotive Devroom

Build Once, Trust Always

Single-Image Secure Boot with barebox

Ahmad Fatoum – a.fatoum@pengutronix.de

https://www.pengutronix.de

About Me

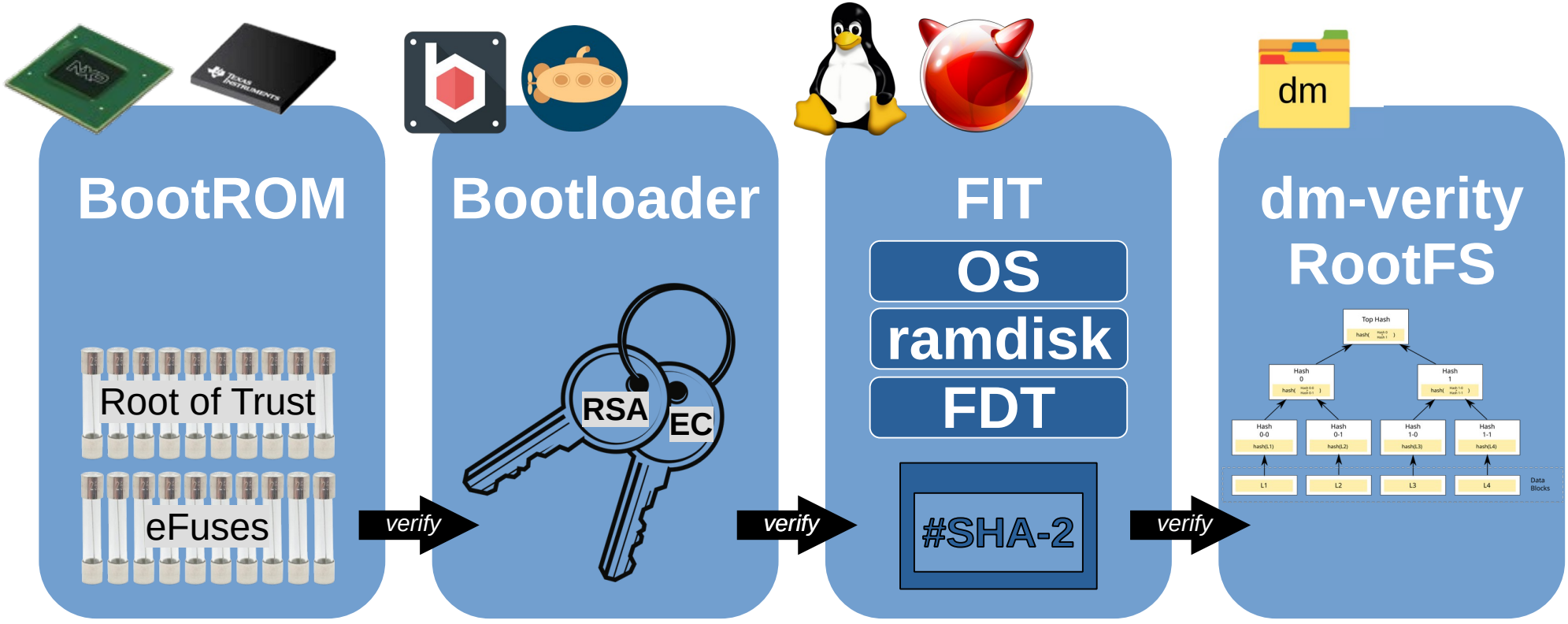
- Ahmad Fatoum
- Pengutronix
- a3f
- a.fatoum@pengutronix.de
- @a3f@fosstodon.org

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

Verified Boot 📈

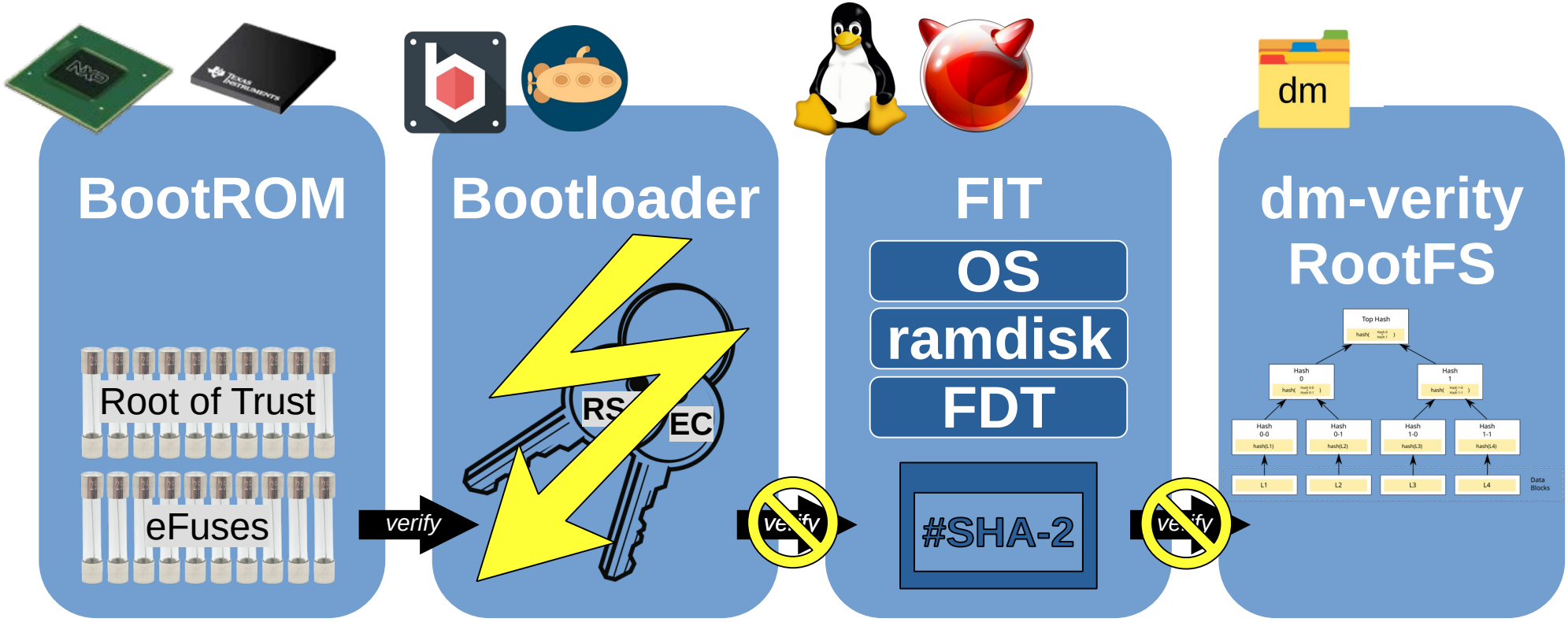
- Most new embedded products secure the boot chain
  - Often motivated by upcoming EU regulation
- For many embedded systems, this takes the form of a verified boot chain

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

Securing the bootloader

- Mostly about restriction:
  - Restrict what it can do
  - Restrict what can be done with it
- Far reaching consequences:
  - Development is more awkward
  - Testing is more complex
  - Maintenance is more work
  - Complexity (and thus risk) is higher
  - Manufacturing and field service is more involved
- Solution required to accommodate the different use cases

Securing the bootloader

- Developers solve problems.

  If the problem is cumbersome development, they will solve that, *but*...
  → State in the field may be become inadequately tested

  → Greatly increases project risk

Device Lifecycle

- 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

Device Life cycle

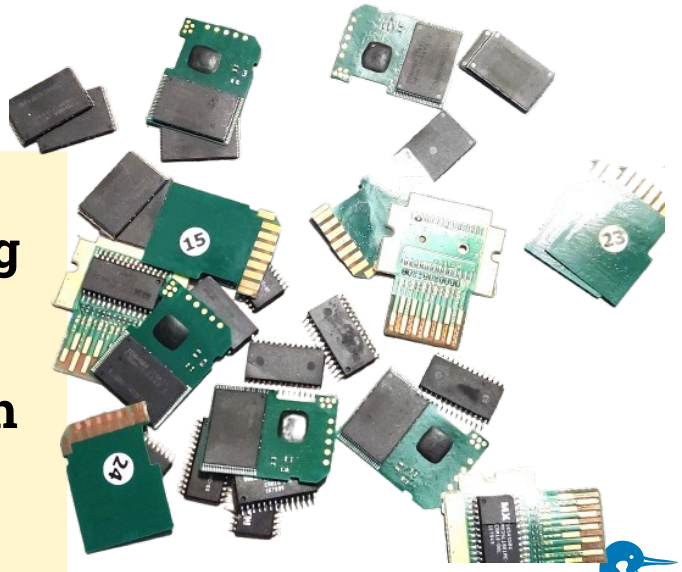
- Security *must* account for life cycle state

  Development ≠ Provisioning ≠ In-Field ≠ RMA

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

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

- Complexity is only shifted around: With life cycle handling in common code, it's more feasible to test it
- In summary, the fewer the images, the better

Fuse-based State Transitions

Problem: OP-TEE RPMB Key Provisioning

- OP-TEE is often provisioned with device-specific certificates
- Certificates need to be sealed with a device-specific key, but:
  - Key is only available after verified boot is activated!
  - Key storage is only possible after eMMC RPMB Key is written
- Problem: Enabling `CFG_RPMB_WRITE_KEY` in the default configuration is a security vulnerability
  - Attacker can replace the eMMC and snoop plain-text key transfer!
- Traditional solution: Multiple images for factory and use in the field.
  - If factory image is leaked, attacker can modify RPMB contents ⚡

Solution?: Fuse-based state transitions

- Always enable `CFG_RPMB_WRITE_KEY`
- Before key write: ensure specific eFuse is not blown
- After key write: Blow the eFuse
  - i.MX On-Chip OTP support: https://github.com/OP-TEE/optee_os/pull/7594
- Write Key is one time operation → Single OP-TEE image!
- https://github.com/OP-TEE/optee_os/pull/7597
- Fruitful upstream discussion
  - Move actual RPMB programming out of OP-TEE
  - Add pseudo TA to retrieve RPMB key in the factory:
  - Still gated behind eFuse not being blown + some replay protection

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

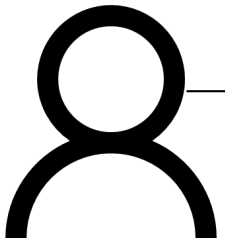
Introducing barebox Security Policies

- Generic code consults the active policy as needed:

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

   /* do stuff */
}
```

- Check directly at the security-sensitive operation (instead of merely marking a console read-only) → More future-proof

barebox Security Policies: Visualized



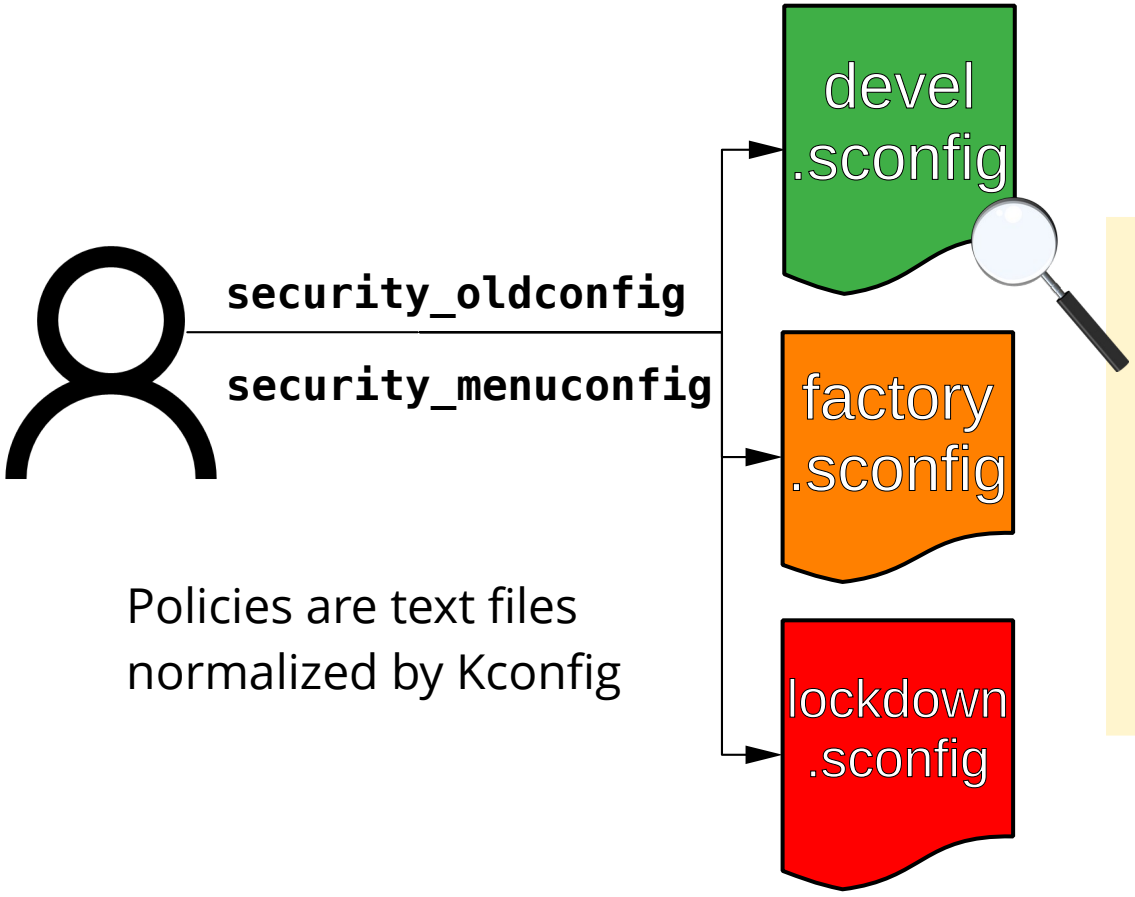
security_oldconfig

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

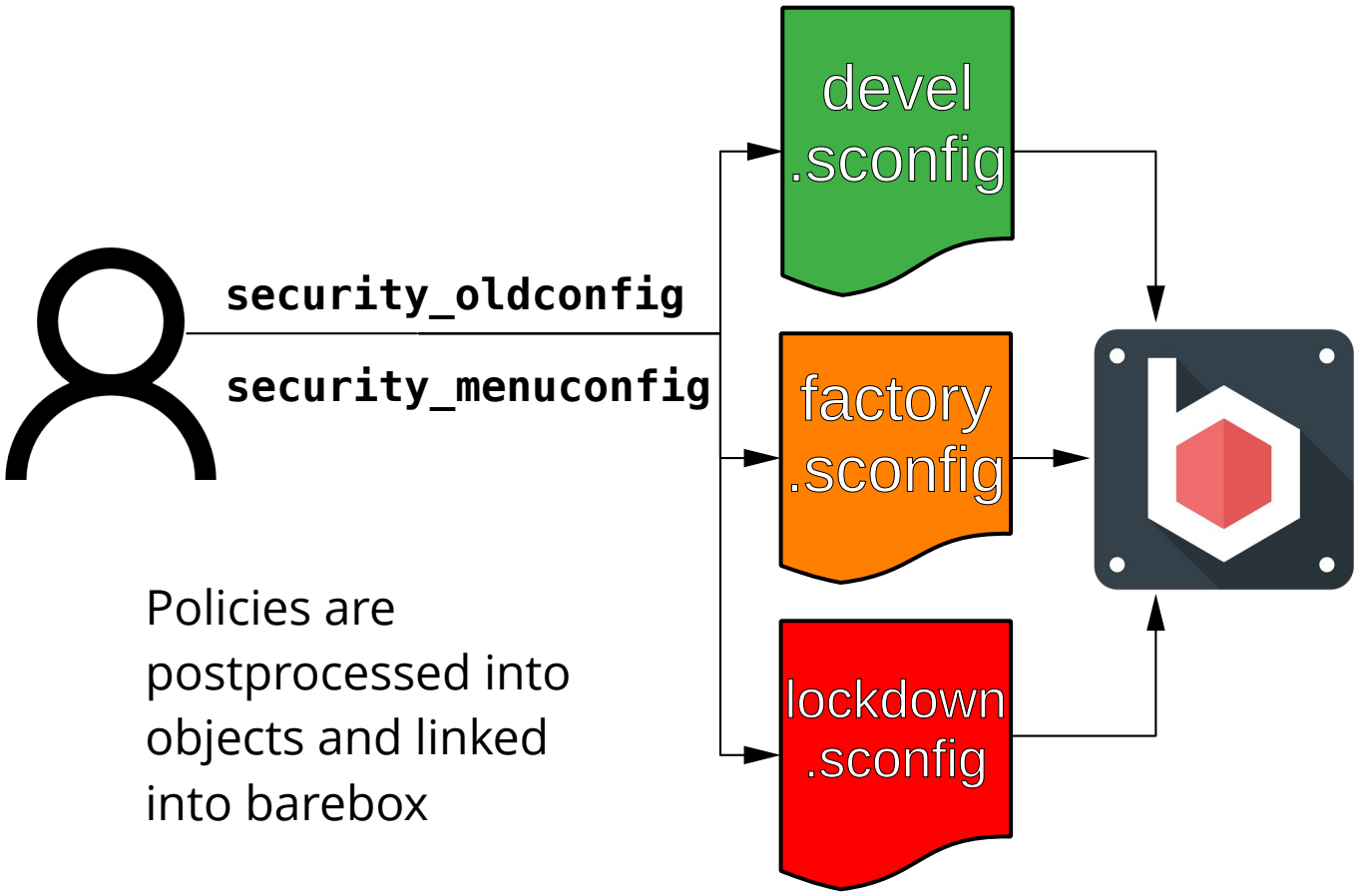
barebox Security Policies: Visualized



Policies are text files
normalized by Kconfig

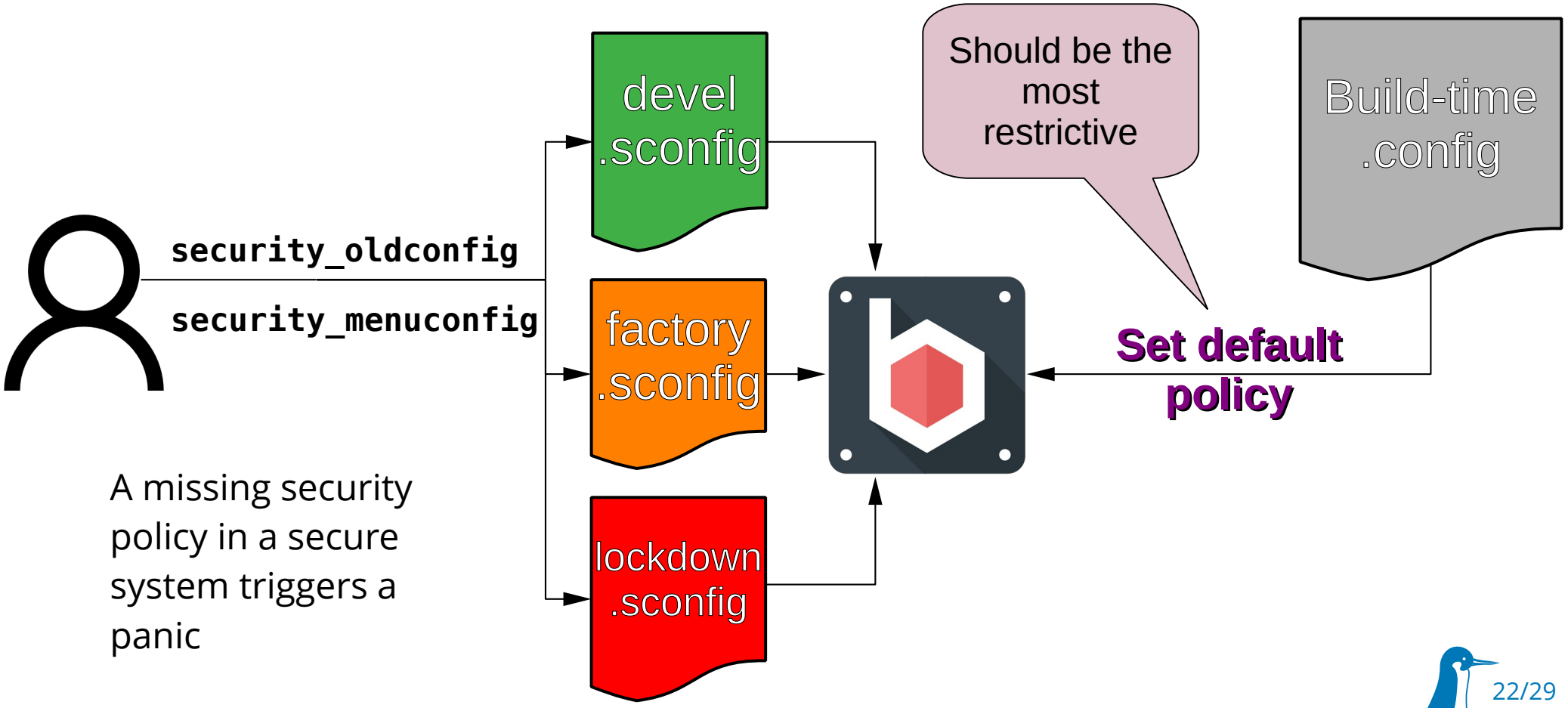
```
SCONFIG_POLICY_NAME="lockdown"
SCONFIG_SECURITY_POLICY_SELECT=y

# 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



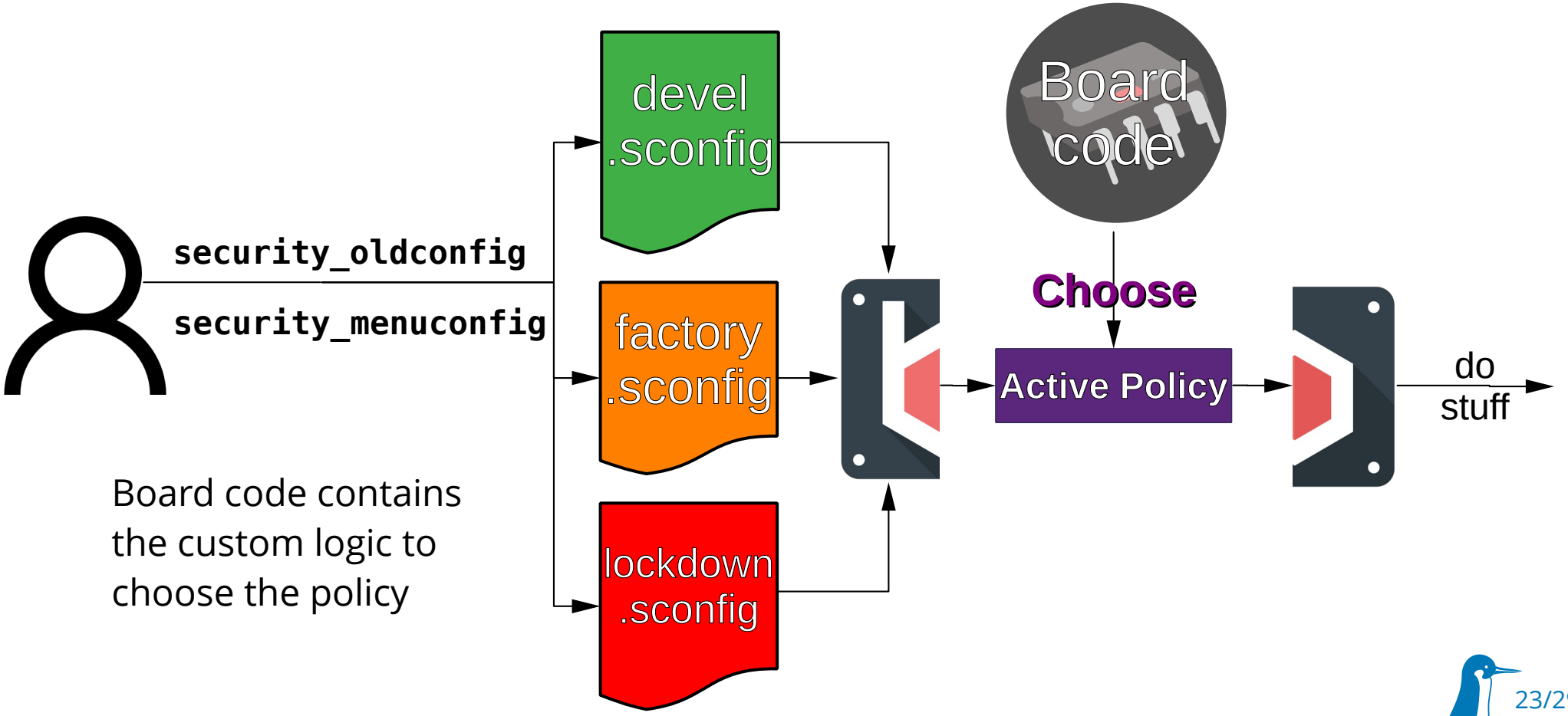
Policies are postprocessed into objects and linked into barebox

barebox Security Policies: Visualized



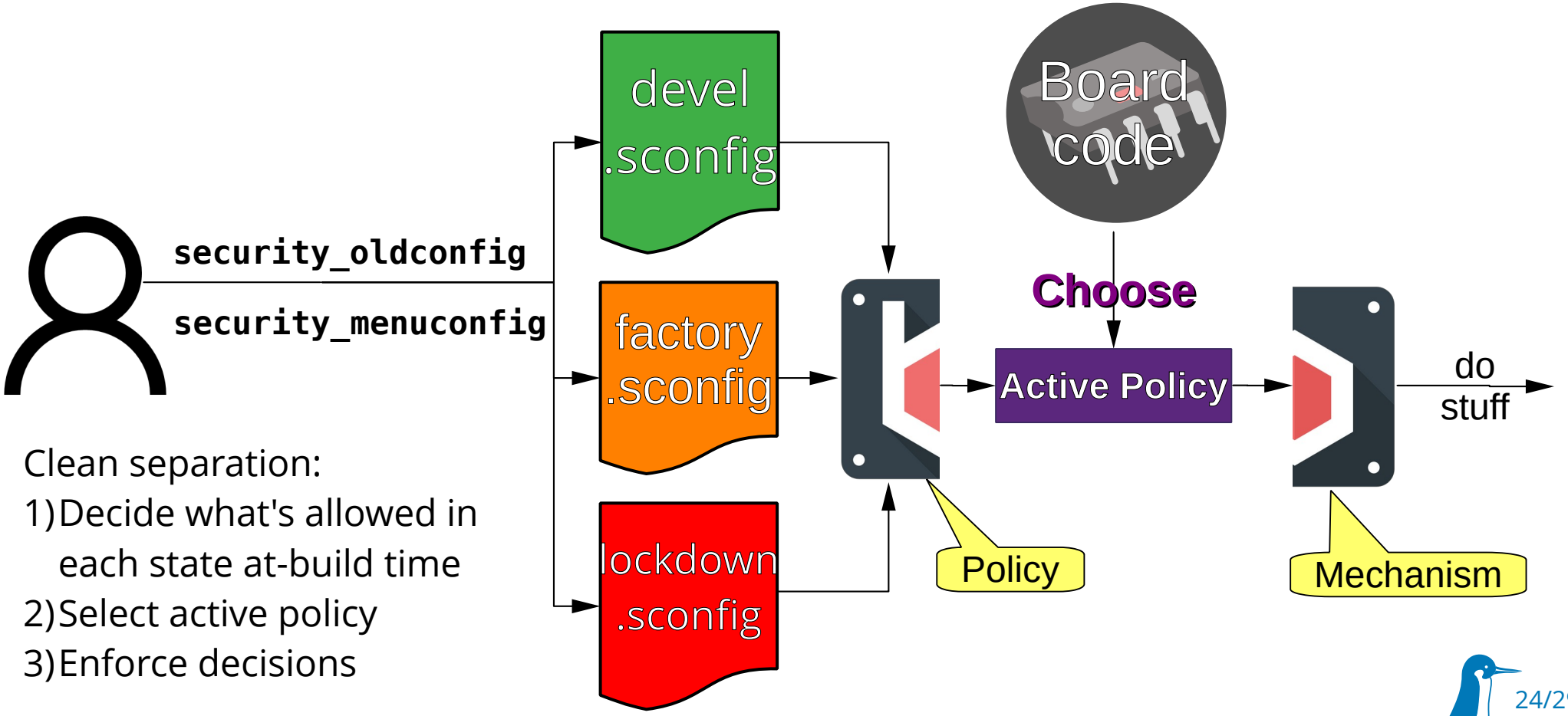
A missing security policy in a secure system triggers a panic

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

Policy selection example

- Board code selects security policy

```
/*
 * 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 */
```

Runtime Unlocking

Unlocking developer devices

```
/*
 * 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
    - See Jonas' talk here in this devroom at 12:00 🔗
- Unlock token must not be transferable across devices
  - A SoC unique ID: `barebox_get_soc_uid_bin()` 🔗
  - A datum in replay-protected memory
    (e.g. Android Verified Boot TA)

Future Outlook

- Generic "System Data" TA
  - Key/Value-Store with rollback protection / write once
  - Securely configure a different security policy with rollback protection
- More memory leaks fixed since introductory talk in August 🔗:
  - Soon submission for oss-fuzz?
- Passing along the security policy to Linux (`ConditionKernelCommandLine=` ?)

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:

Questions?