RAUC

Behind the Scenes of an Update Framework

ELC Europe 2019

Enrico Jörns – e.joerns@pengutronix.de
About Me

- Embedded software developer
- RAUC co-maintainer
- At Pengutronix since 2014

![Pengutronix](image)

- Embedded Linux consulting & support since 2001
- > 5000 patches in Linux kernel
Introduction
Updating – Big Picture
Updating – Big Picture
Updating device. Do not turn off!
Fail-Safe Updating

A: Active (running) system
B: Inactive system

Bootloader
Fail-Safe Updating

- Deactivate partition to update

Bootloader
Fail-Safe Updating

- Deactivate partition to update
- Write update(s) to disk

Critical Operation!

Bootloader
Fail-Safe Updating

- Deactivate partition to update
- Write update(s) to disk
  Critical Operation!
- Update fully completed + verified, etc.

Bootloader

A

B

Verified
Fail-Safe Updating

- Deactivate partition to update
- Write update(s) to disk
- Critical Operation!
- Update fully completed + verified, etc.
- Activate updated slot
Overview Over FOSS Update Frameworks

- Image-based
  - MENDER
  - RAUC
  - swupdate

- Others
  - OSTree
  - balena
about:rauc

- **Name:** „Robust Auto-Update Controller“
- **Subject:** FOSS update framework
- **Age:** Started in 2015, project-driven
- **License:** LGPL-2.0
- **Version:** 1.2
- **Community:** ~50 contributors (1450 commits)
- **Recent:** Ongoing development / fixes / adaptations
RAUC – Design Goals

- Generic declarative framework
  - A+B, A+recovery, A+B+recovery
  - Platforms / bootloaders
  - Application integration
- Limited complexity
- Security (update verification)
- Robust design
  - Error Handling, standard libraries, subprocess calls
RAUC – Host + Target Tool

- host tool
- compile
- target service / tool
RAUC – Code & Service Architecture

- C, automake
- Utility library → GLib
- D-Bus (optional)
- Subprocess calls for robustness

Service architecture
Slots – Configuring Redundancy Setup

device view

RAUC slot view

configuration (in rootfs)
RAUC – Slot Handling

update bundle

rootfs image

appfs image

rescue.0

rootfs.0

appfs.0

rootfs.1

appfs.1

bootloader.0

inactive

active

booted

target
Service determines target slots
Booted Slot Detection

- Preferably explicit

```
... console=ttyS0,115200 rauc.slot=system0 root=...
```
→ must match bootname

- Alternatively via `root=`

```
... console=ttyS0,115200 root=UUID=1b77ad0b-... ...
```
→ must be detectable in mounted file systems
$ rauc status

[...]

== Slot States ==

[bootloader.0] (/dev/mtd0, raw, inactive)

○ [rootfs.0] (/dev/sda1, raw, booted)
  
  bootname: system1
  boot status: good

  [appfs.0] (/dev/sda3, raw, active)

• [rootfs.1] (/dev/sda2, raw, inactive)
  
  bootname: system0
  boot status: good

  [appfs.1] (/dev/sda4, raw, inactive)
A bundle describes the intended target state of the system.

- **SquashFS**
  - Mountable!
  - Compression
- **Signature (CMS)**
  - Standard X.509 PKI
  - Verification on Target
Manifest – System Configuration

[update]
compatible=MyProduct2000
version=2019.10-4
build=20190228134503

[image.rootfs]
filename=rootfs.ext4
size=419430400
sha256=b14c1457dc1046...

[system]
compatible=MyProduct2000
bootloader=barebox

[keyring]
path=/etc/rauc/keyring.pem

[slot.rootfs.0]
device=/dev/sda1
type=ext4
bootname=system0

[slot.rootfs.1]
device=/dev/sda2
type=ext4
bootname=system1

...
HOW to install – Update Handler

```
slot type + image type (filename) = chosen handler
```

- ext4 + ext4 = raw copy
- ubifs + tar.xz = mkfs.ext4
- ubifs + ubifs = mount
- nand + img = tar xf
- vfat + ... = umount

system.conf  Manifest
RAUC Bootloader Interaction

- **Installation**
  (Atomic action)

- **Boot acknowledge**
  (fallback handling)
Customizing The Update

- System Configuration
- Hooks
  - In update bundle
- Handlers
  - In system
- Full custom handler
  - Only use signature verification
RAUC – Authentication

- RAUC host tool signs Bundle
- Verification on target
- OpenSSL 1.x
- X.509 crypto / CMS
- Self-signed to full PKI
- Key revocation and replacement
RAUC Signing Features – X.509 PKI

- Bundle
  - devel signature
  - release signature
  - resigning
  - signer A
  - intermediate 1
  - intermediate 2
  - multiple signer
  - HSM (PKCS#11) support
RAUC – Advanced Topics
RAUC And Verified Boot

- **dm-verity**
  - Build System
    - image
      - hash tree
    - install
    - block device
      - r/o
    - Target
      - r/w

- **dm-integrity**
  - Build System
    - tar
  - extract
    - ext4
    - dm-integrity
    - journal tags
    - block device
  - Target
    - r/w
Built-In Bootloader Updates

- Single point of failure
  → Critical Component
- No Fallback, but atomicity!
- MBR
- If supported by ROM loader / storage:
  - eMMC
  - NAND (i.MX6)
Atomic Bootloader Updates – eMMC

- built-in dual boot partitions
- extCSD register selects active one

```bash
... [bootloader.0]
type=boot-emmc
device=/dev/mmcblk0
...```
Atomic Bootloader Updates – MBR

- ROM loader boots from first MBR partition
- Switch between redundant partition regions

```...
[bootloader.0]  
type=boot-mbr-switch  
device=/dev/sda1  
...```
Streaming and Delta Updates – casync

- Image updates over Network
  - Too large (slow connection)
  - Temporary storage required

→ delta updates
→ avoid reinventing the wheel

“casync (content-addressable synchronisation) is a Linux software utility designed to distribute frequently-updated file system images over the Internet.”

[Wikipedia]
casync – RAUC
Usage and Integration
Integration – Required Components

host
- keypair
- host tool
- utilities

target
- keyring
- utilities
- target service
- system config
Integration & Ecosystem

- Linux build system integration
  - Yocto (meta-rauc)
  - PTXdist
  - Buildroot
- Into Application: D-Bus
- Example projects like rauc-hawkbit
Thank you!

Questions?
References

RAUC system update documentation:
https://rauc.readthedocs.io/en/latest/

RAUC on GitHub:
https://github.com/rauc/rauc

meta-rauc:
https://github.com/rauc/meta-rauc

casync:
https://github.com/systemd/casync