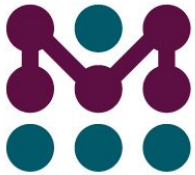
Software Updates for Connected Devices

Key Considerations



MENDER.io

Deploy Software Updates for Linux Devices

Eystein Stenberg
CTO
Mender.io

Who am I

- Eystein Stenberg
  - 7 years in systems security management
  - M. Sc., Computer Science, Cryptography
  - eystein@mender.io
- Mender.io
  - Over-the-air updater for Linux, Yocto Project
  - Open source (Apache License, v2)
  - Dual A/B rootfs layout (client)
  - Remote deployment management (server)
  - Under active development

Connected devices must be remotely updatable

- There *will* be **bugs**, **vulnerabilities**
  - 1-25 per 1000 lines of code*
- … and new **features**
- … after device is **deployed to the field**

Fiat Chrysler recalls 1.4 million cars after Jeep hack

24 July 2015 | Technology

Critical- and high-severity security bugs in Linux

Source: Ars Technica

**Source: Steve McConnell, Code Complete*

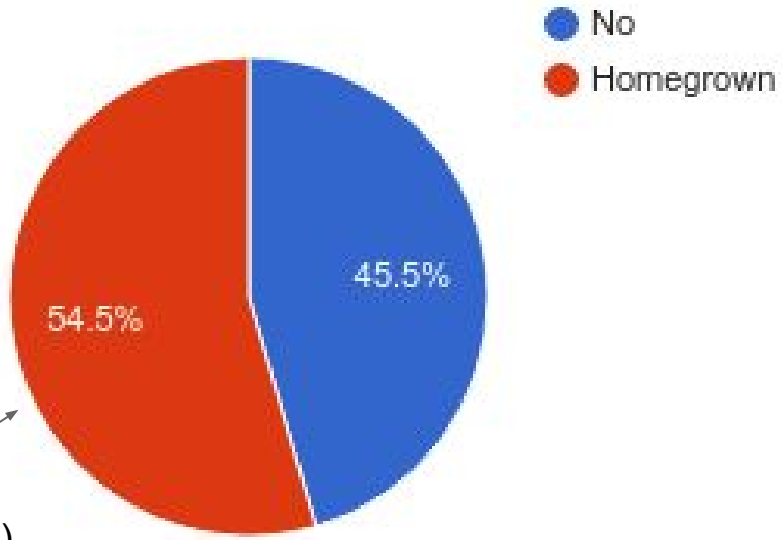
Reuse open source tools for remote updater

- Most companies write their own homegrown updater from scratch (!)
  - More on this later
- Existing free and **open source tools** can help you
- Save time and avoid stress (where you can)
  - Initial development
    - It looks so easy! **Is it easy?**
  - Ongoing maintenance
    - 5 years? 10? 20?
    - 1 product? 5? 10?

Session overview

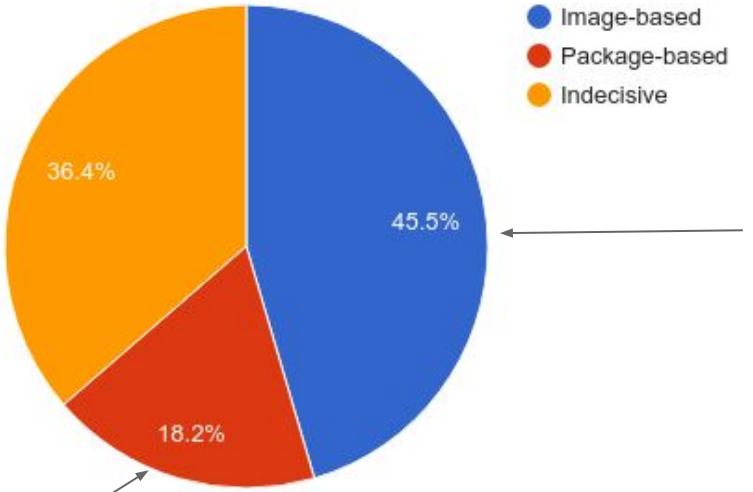
1. Survey: state of updating embedded software today
   - 30 interviews
2. Environment and criteria for embedded updater
3. Solution strategies for updating embedded devices

Do you deploy updates today? How?



Includes one-by-one (standalone)

Image-based or package-based deployment?



- “Atomic”
- “Consistent”

- “Fast installation”
- “Easy to develop”
- “Uses less bandwidth”

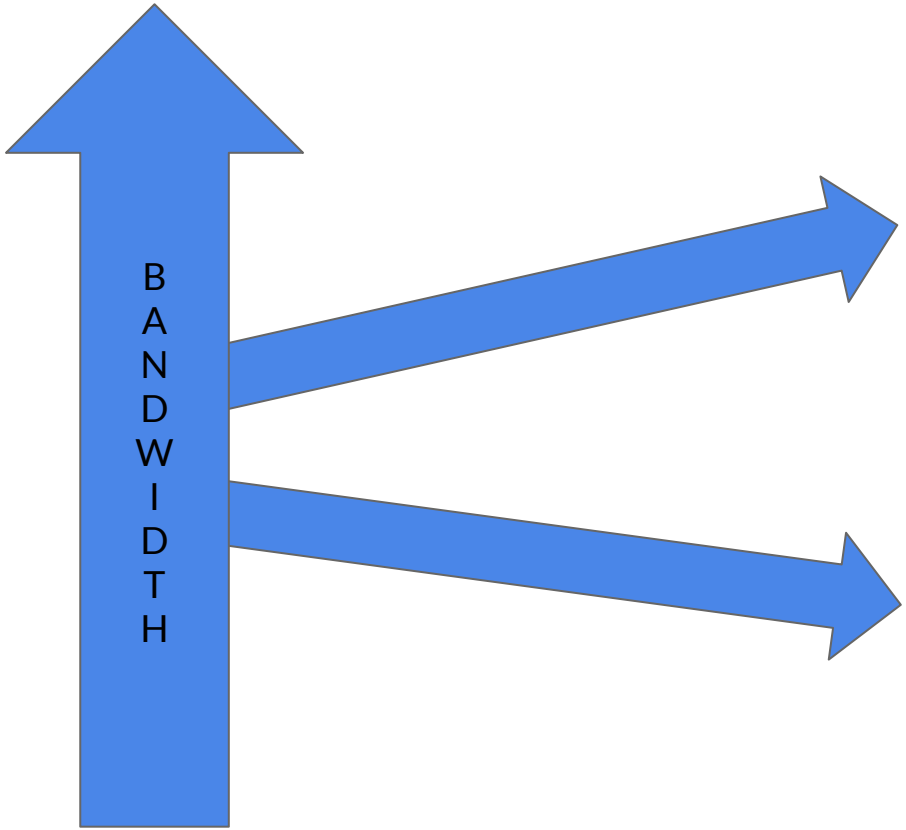
Development time and frequency of use

- Q: How long did you spend for **initial development** for **homegrown** updater?
  - A: 3-6 months
  - Maintenance time comes in addition
- Q: How **frequently** do you deploy remote updates (if you can)?
  - A: 6 times / year (per product)
- Bottom line: ways to go
  - You are for sure not far behind average
  - Interest has picked up, so heading in right direction (see other sessions)
  - Connectivity of embedded devices (“IoT”) is biggest driver

The embedded environment

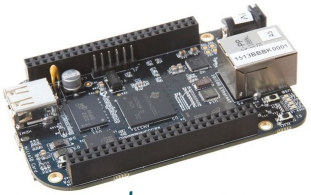
- Remote
  - Expensive to reach physically
- Long expected lifetime
  - 5 - 10 years
- Unreliable power
  - Battery
  - Suddenly unplugged
- Unreliable network
  - Intermittent connectivity
  - Low bandwidth
  - Insecure

Network requirements are *different than for smartphone*



$700

- Expensive
- High data speed
- 3G, 4G, 5G, wifi



$30

- Low cost
  - Hardware
  - Data transfer
- Small size
- Low data speed
- High connectivity

Key criteria for embedded updates

1. Robust and secure
2. Integrates with existing environments
3. Easy to get started
4. Bandwidth consumption
5. Downtime during update

1. Robust and secure

- Must handle power and network loss at any time

- **Atomic** installation; definition:
  - An update is always either **completed fully, or not at all**
  - **No software component** (besides the updater) ever **sees a partially installed** update

- **Consistent** deployments across devices
  - test = prod

- Sanity check **after** update
  - Ensure we can do **another update** (“it boots” is not enough)
  - Custom checks (can we reach servers, are services running, etc.)

- Ensure **authenticity** of update

2. Integrates with existing environments

- Few projects have the luxury of “starting from scratch”, but have existing
  - Development tools, build/test environments
  - Hardware (storage types, network, etc.)
  - Operating systems (Yocto Project based, FreeRTOS, etc.)
  - Devices in the field

- Standalone mode
  - E.g. supports update-via-USB/SD
  - Transition to remote updates may take time (no connectivity today?)

- Extensible
  - Custom update actions
  - Sanity checks, pre- & post-install scripts
  - Custom installers

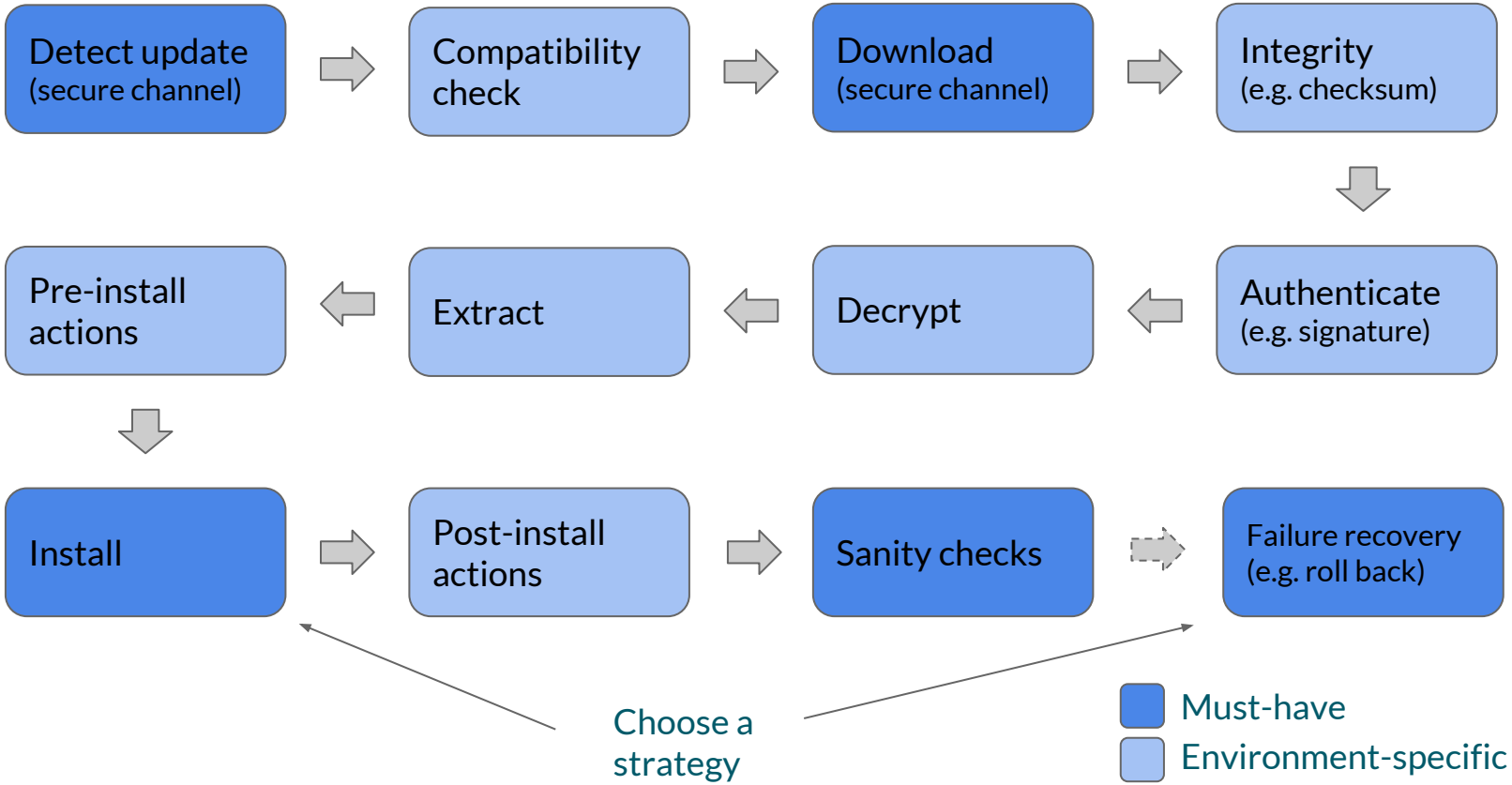
3. Easy to get started

- How long does it take to test out (e.g. on reference board)?
  - Need to decide if it is suitable or not
  - Too difficult: decide to build homegrown?
- High quality? Well tested? How do you know?
  - Test reports
  - Successful adoption by others
- Is it well documented?

4 & 5. Bandwidth & downtime during update

- “Lower is better” for both these metrics...
- … but requirements vary widely between systems
- Safety critical system - needs to be always up?
  - Maintenance windows?
  - How long and frequent?
- Which network is used?
  - Cable, wifi
  - Cellular
  - Sigfox

Generic embedded updater workflow

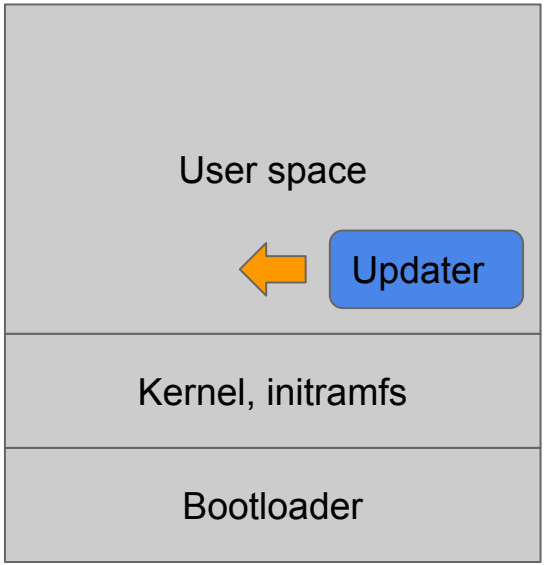


Generic criteria - any installer strategy

- No matter *how you install* (up next), you can achieve these
- Typically tool-specific

- Sanity checking (1. Robust and secure)
- Authenticity (1. Robust and secure)
- Standalone (2. Integration)
- Extensibility (2. Integration)
- Time to test (3. Easy to get started)
- Quality (3. Easy to get started)
- Documentation (3. Easy to get started)

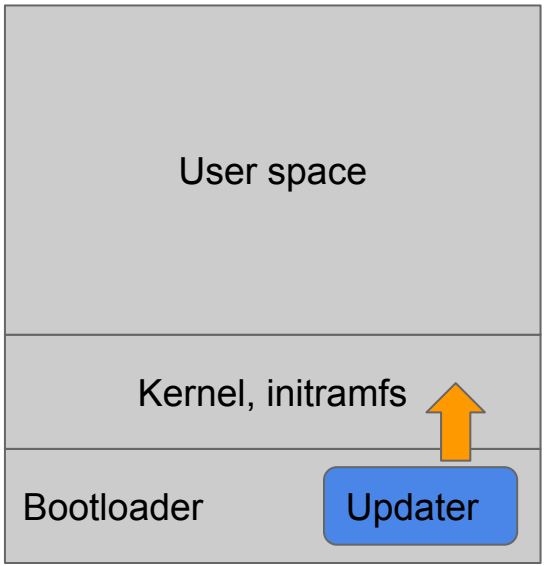
Installer strategy 1: run-time installation



- Updater deploys to running environment
  - Package managers (ipkg, rpm, deb...)
  - OSTree
  - Many homegrown (tar.gz)

- Robustness is hard
  - Atomicity: Hard or impossible
  - Consistency (dev=test): Hard
- Integrates well
  - May already have packages
  - Some userspace tools
- Low bandwidth use (<1mb)
- Short downtime (seconds)

Installer strategy 2: boot to maintenance mode

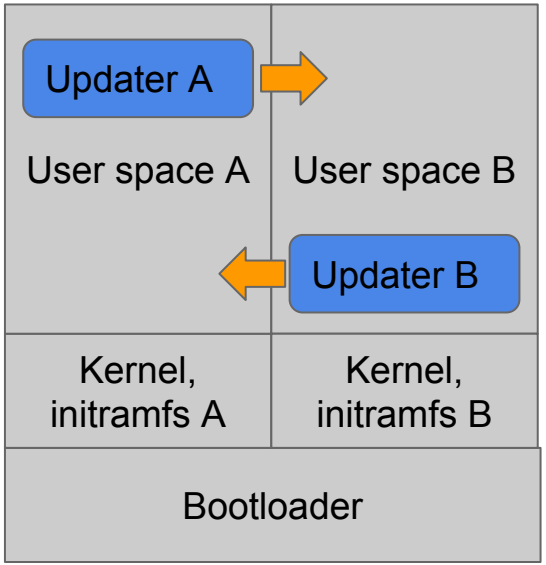


- Updater deploys “up the stack” while running in bootloader
  - Used in older Androids (before ‘N’)
  - “Rescue environment” common in embedded

- Robustness is hard
  - Not atomic (can get partial update)
  - Consistent on success (image)
- Integrates fairly well
  - Bootloader features & intelligence
- High bandwidth use*
  - Whole image
- Long downtime
  - Whole image install
  - 2 reboots

*Can mitigate: compressed/delta

Installer strategy 3: dual A/B rootfs layout

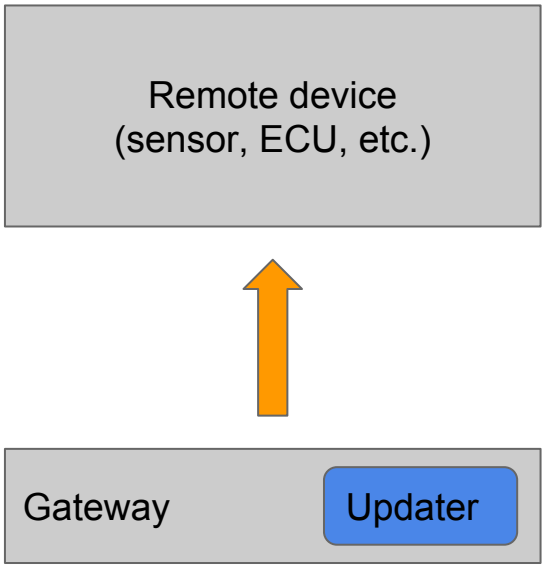


- Updater deploys to inactive partition, then reboots into it
  - Used in newer Androids ('N' and later)
  - Common in "mid/high-end" embedded

- Very robust
  - Fully atomic and consistent
- Integrates fairly well
  - OS, kernel, apps unchanged
  - Needs bootloader "flip" support
  - Partition layout, requires 2x rootfs storage
- High bandwidth use*
  - Whole image
- Fairly short downtime (minute)
  - 1 reboot

*Can mitigate: compressed/delta

Installer strategy 4: proxy



- Updater deploys to remote system
  - Used on smaller devices (sensors, ECUs, etc.), such as in Smart Home or Automotive
  - Requires intelligent gateway to manage

- Slightly different scenario
  - Smaller devices (no client)
  - Complements the others

- Suited for closeby installations only, not internet
  - Robustness (e.g. connection/power loss)
  - Security

Comparison of installer strategies

	1. Run-time installation	2. Boot to maintenance mode	3. Dual A/B rootfs
Atomic			
Consistent			
Workflow integration			
Bandwidth			
Downtime			

Remote deployment management

- Efficiently deploy to remote devices
  - It's 2016!
- Server that connects to & manages all client updaters
- Group devices
  - E.g. by customer
- Campaign management
- Reporting

Ensure your devices can be updated remotely

- Choose installer strategy that fits your environment the best
- Reuse open source tools where possible
  - Writing from scratch not as easy as it seems (for production use)
- Mender.io implements dual A/B rootfs layout
  - Try in 10 minutes on mender.io

Rest assured you can fix that undiscovered bug!