SECURE BY DESIGN SERIES

Secure Software Updates: Designing OTA Updates for secure embedded Linux systems

Maciej Halasz
Timesys Corporation

SECURE CONNECTIONS
FOR A SMARTER WORLD

EXTERNAL

Agenda

Software updates – Why do we need them?

What to update?

- Architecture for OTA

Linux software update options

- swupdate
- ostree
- containers

Security of a software update

How to manage secure software update?

- Scripts
- hawkBit

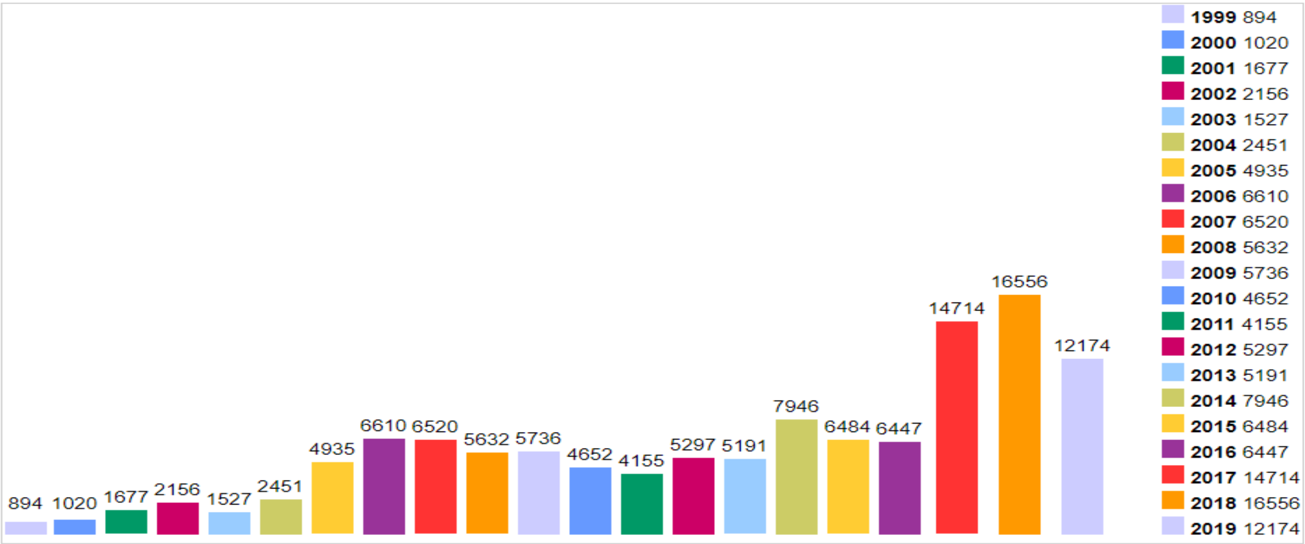
Software Updates – Why do we need them?

NXP | SECURE CONNECTIONS FOR A SMARTER WORLD

Why software updates?

- Designing secure devices does not end with the initially deployed software
- Number of reported software vulnerabilities is growing rapidly
- End users and companies are more aware and more concerned about privacy and data protection
- Security is a MUST HAVE requirement for most compliance and standard guidelines
- Software updates are a way to manage:
  - New features
  - Bug fixes
  - Security fixes

Vulnerabilities By Year



Embedded Device Environment

- Power
  - Often unreliable
    - Outages
    - Battery depletion
- Unreliable connectivity
  - Mobile connections (WiFi, BLE, LTE)
  - Low bandwidth
- Often times remote
  - Limited access
- Lifespan
  - 5-20 years, depending on device type

Server update side is more under control

- Secure
- Reliable Power and Network
- Easy human intervention

What to update?

SECURE CONNECTIONS
FOR A SMARTER WORLD

What to update?

- File (Application, user space component or Linux kernel)
  - Typically managed by a custom process, developed specifically for a product/company. Leverages open source techniques
- Feature (package)
  - Package managers such as apt-get or yum work great for servers but are an overkill for embedded devices
- Entire filesystem, complete image
  - One of the most common approaches in embedded space. Can be implemented with one of FOSS solutions
- Entire filesystem, atomic, differential update
  - Commonly used, does a clever atomic update of group of files
  - Used in situations where bandwidth is limited
- Container
  - Operating system and applications can be deployed in self contained files

Requirements for software updates

- Secure
  - Mechanism cannot become an attack vector for a device
- Atomic
  - An update must be installed completely or not performed at all
- Fail-safe
  - In the event of a failed update, fallback to last known good state
- Complete
  - Capable of updating all software layers
- Frequency
  - More frequent updates may drive selection of the update method
- Speed of an update
  - Just like with frequency, the time in which we need a device updated may drive selection of the update method

Linux software update options

Local Update

- Software update with local presence at the device
- Pros:
  - Can recover device easily
  - Can ensure it boots correctly into desired image
- Cons:
  - Not very scalable
  - Time consuming process
  - Requires physical access to a device
  - Requires certain interfaces to be available e.g.: USB (security attack vector)

Over-the-Air (OTA) Update

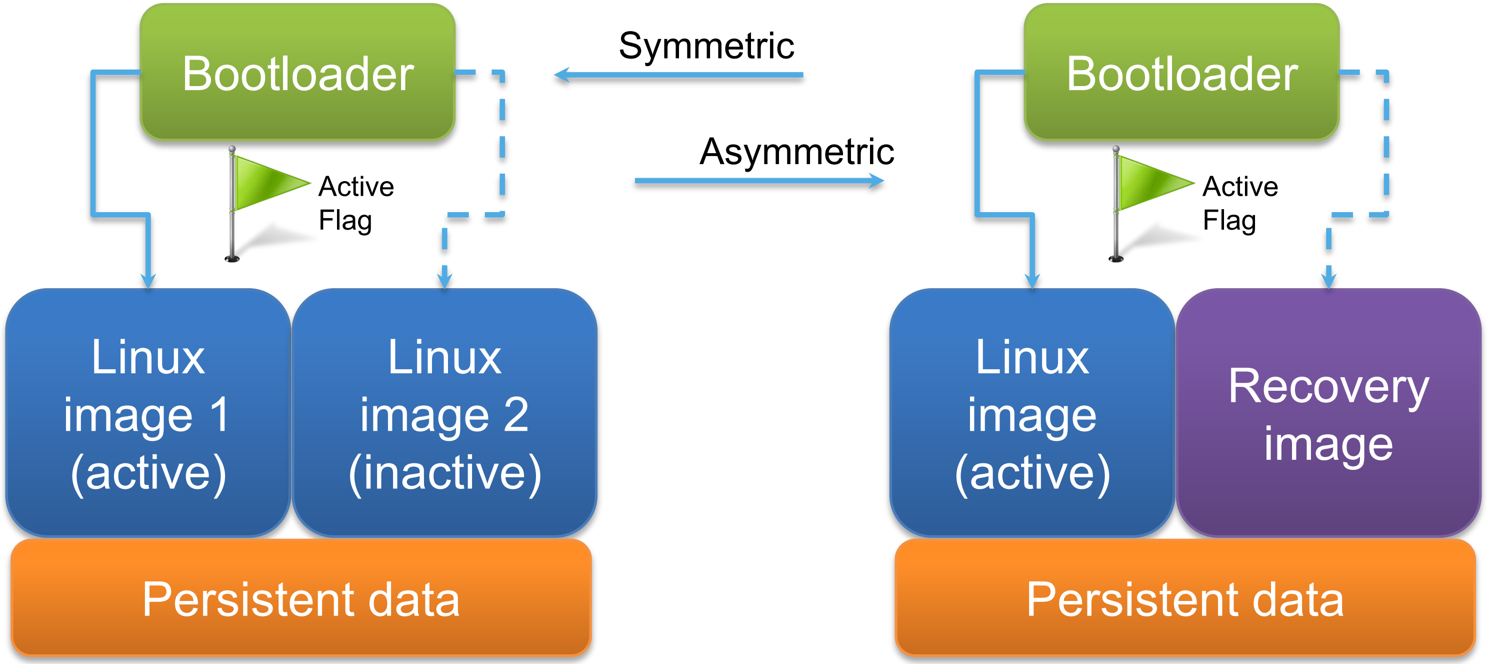
- Software update is done remotely from a provisioning server
- Pros:
  - Does not require physical access to the device
  - Scalable
  - Only network communication required for an update
- Cons:
  - Limited remote troubleshooting
  - Requires elaborate server system to manage remote devices

Update with a fallback

- Update mechanism
  - Symmetric: Requires double the storage space for the update
  - Asymmetric: Update performed in place by booting into initramfs
  - Reboot required
  - Fallback feature
- Example technologies for image updates
  - swupdate
  - Android

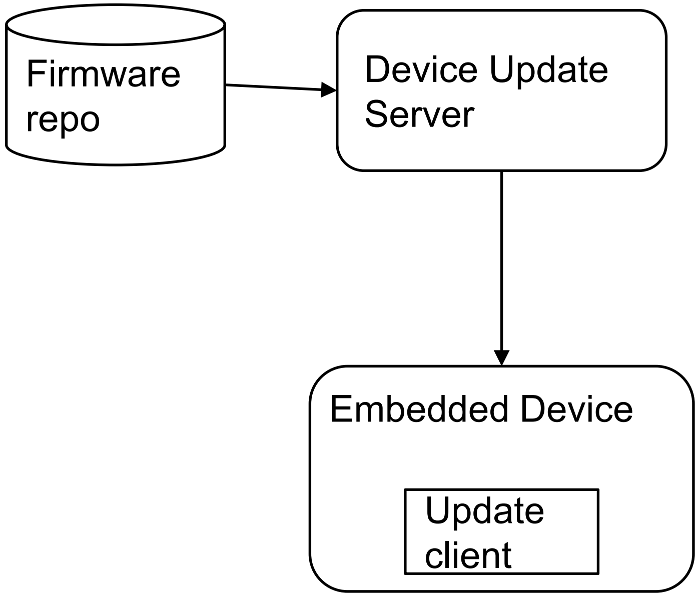
Update Process:

1. Boot into active Linux image 1
2. Receive/Install Linux image 2
3. Toggle Active flag/Reboot
4. If reboot fails fallback to previous active image (watchdog/heartbeat)



Update agent/client

- Script or application deployed on a device that manages the update process
- Applicable to offline and OTA update processes
- Tasks performed depend on the server side framework used
- Example tasks
  - Authenticate server connection
  - Download select update software
  - Perform local security checks
  - Perform local update
  - Toggle the boot flag

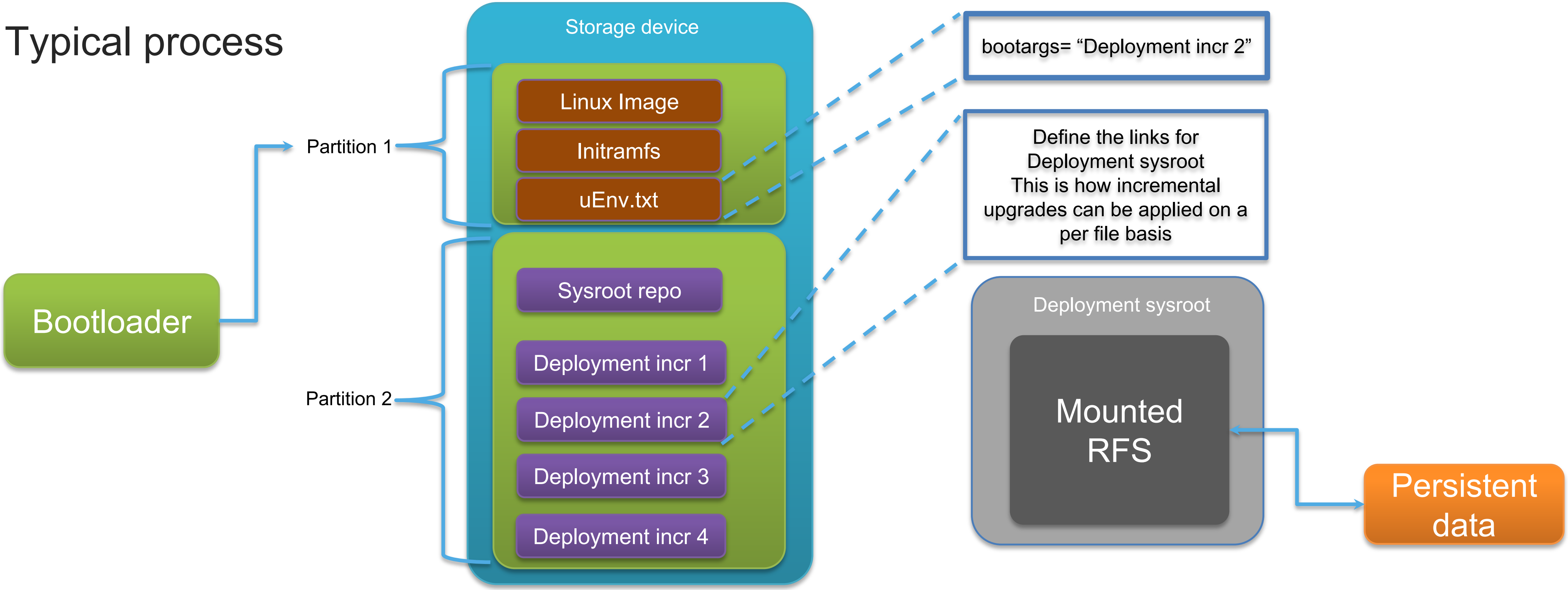


Atomic image update approach

- Applies set of rules to update Linux image in place
- Update deltas are per modified files
- Allows for smaller data transfers
- Fast update
- Incremental atomic upgrades that can be quickly deployed or rolled back on demand
- Example technologies for atomic differential update
  - OSTree
  - swupd

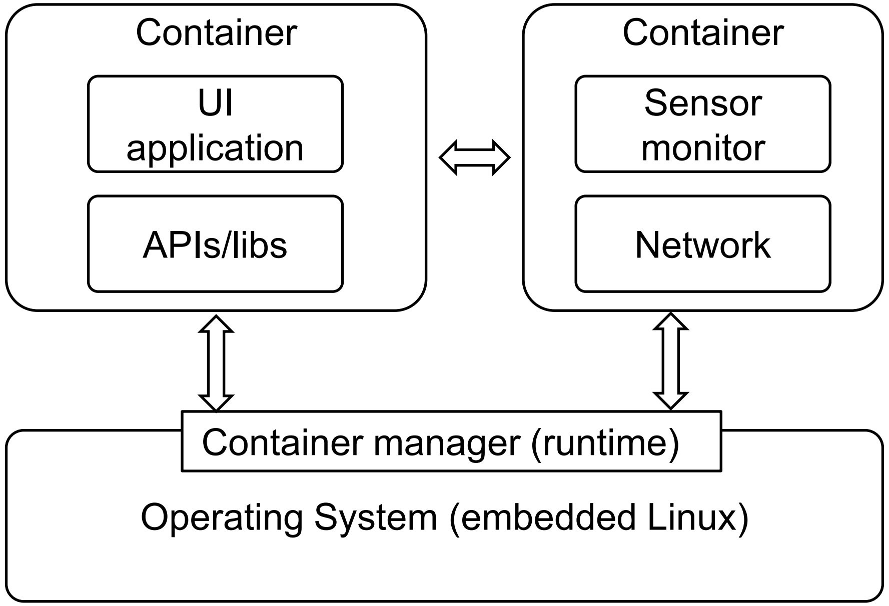
Atomic differential update approach

- Typical process



Containers

- Container allows for separation of applications from OS
- Each application/feature can be self contained and independent of other applications
  - Can run Qt5.2 and Qt5.12 apps on the same device
- Can limit container resource access
  - CPU usage, Memory usage, Network
- Requires runtime container manager to be installed in OS
- Updates of application containers are atomic and recoverable



OTA Areas of Concern

- Roll-back
  - What happens when we lose power during an active update?
- Firmware Security
  - Is what we downloaded coming from our company?
  - Are there any “trojan horses”?
- Scale
  - How do I manage firmware for many device types and many devices?
  - How do I roll out a firmware update in large volume devices?
- Monitor
  - How do I know which devices need an update?
  - How do I know which devices failed and need recovery?

Select Software Update Frameworks

SECURE CONNECTIONS
FOR A SMARTER WORLD

swupdate (1)

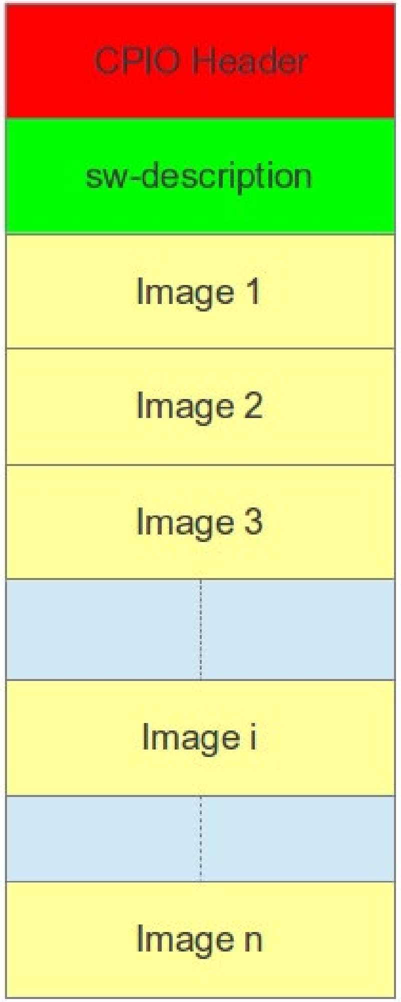
- Symmetric or Asymmetric image update framework

  https://github.com/sbabic/swupdate

- Written in C
- GPLv2 license
- Supports signed images, Local/OTA updates, and U-Boot
- Has an active development community
- Used in commercial setting
  - e.g.: Siemens (https://openiotelceurope2016.sched.com/event/7rrA)
  - e.g.: Bosch (https://docs.bosch-iot-rollouts.com/documentation/introduction/ecosystem.html)

swupdate (2)

- Updates delivered in simple CPIO archive (.SWU)
- swupdate tools must be installed in a root filesystem
- Each individual image is described in sw-description and integrity is validated with a SHA256 hash
- Handler plugins implement the details of how each described image is handled
  - U-Boot env update
  - NOR, NAND, UBI partition and write
  - MMC/SD/eMMC partition and write
- Uses menuconfig for configuration
- Supports REST interface to Hawkbit server for remote update



```
software =
{
        version = “3.5.1”;
        ventilator_iMX8 = {
                hardware-compatibility : [ “REV7” ];
                images : (
                           {
                                filename = “yocto-rootfs.ext2.gz”;
                                device = “/dev/mmcblk2p2”;
                                type = “raw”;
                                sha256 =
“8e760f8378c273894adf78bba983ced094039cc8746272f484909aca84728b7”
                                compressed = true;
                           }
                );
                scripts : (
                           {
                                filename = “device_update.sh”;
                                type = “ shellscript “;
                                sha256 =
“82a8738928cf8382908f928bba983c63729dcc8378746272f484909aca83783c”
                           }
                );
        };
}
```

Main block

Hardware association

Handler for rfs image

Handler for scripts

Example Yocto recipe (ventilator-iMX8-swu-image.bb)

```
# Copyright (C) 2015 Unknown User <unknown@user.org>
# Released under the MIT license (see COPYING.MIT for the terms)

DESCRIPTION = "Example Compound image for Ventilator i.MX8 board "
SECTION = ""

# Note: sw-description is mandatory
SRC_URI_ventilator_iMX8 = "file://sw-description \
        "
inherit swupdate

LICENSE = "MIT"
LIC_FILES_CHKSUM = "file://${COREBASE}/LICENSE;md5=4d92cd373abda3937c2bc47fbc49d690 \
                    file://${COREBASE}/meta/COPYING.MIT;md5=3da9cfbcb788c80a0384361b4de20420"

# IMAGE_DEPENDS: list of Yocto images that contains a root filesystem
# it will be ensured they are built before creating swupdate image
IMAGE_DEPENDS = ""

# SWUPDATE_IMAGES: list of images that will be part of the compound image
# the list can have any binaries - images must be in the DEPLOY directory
SWUPDATE_IMAGES = " \
                core-image-full-cmdline \
                "

# Images can have multiple formats - define which image must be
# taken to be put in the compound image
SWUPDATE_IMAGES_FSTYPES[core-image-full-cmdline] = ".ext4"

COMPATIBLE = "ventilator_iMX8"
```

swupdate & Yocto Project

- There is a ready-made metalayer for a Yocto Project based Linux BSP
- Called meta-swupdate, it can be downloaded from:

  https://github.com/sbabic/meta-swupdate

- By default it is set to generate asynchronous recovery deployment image

```
$ bitbake swupdate       - image
```

- Standard Yocto build images are generated in tmp/deploy/
- ext2.gz root filesystem
- A custom update image (.swu) can be generated with a custom image recipe which uses desired sw-description file

```
$ bitbake ventilator          - iMX8 - swu- image
```

swupdate command

Local Update

- Run a script every time a USB drive is mounted
- Commands to be executed:

```
        $swupdate    - i <name_of_update>
or
        $swupdate    - i <name_of_update>        - k
<pubkey>
```

- Assumptions:
  - Software update image .SWU is available (downloaded)
- Example:

```
        $ mount /dev/sda1 /mnt/
        $ swupdate     - i /mnt/ventilator           - i.mx8  -
swu- image.swu
        $ reboot
```

Over-the-Air (OTA) Update

- Download file from an HTTP server

```
$ swupdate     - k <pubkey>     - u <URL>
```

- Get images from a HAWKBIT server

OSTree (1)

- Incremental atomic differential upgrade mechanism

  https://ostree.readthedocs.io

- LGPLv2+ licensed
- Often referred to as “git for operating system binaries”
- Stores data in a “git-like” object repo to record and deploy complete file system trees using binary deltas
- Yocto Project infrastructure provided under meta-updater
- Deployment sysroot as an OSTree commit
- Make bootloader and initramfs work together to boot the deployment
- Projects that leverage OSTree:
  - Qt OTA
  - Automotive Grade Linux (AGL)

OSTree (2)

- Structure on the target
  - /ostree/repo
  - /ostree/deploy
    - Multiple deployments stored here
  - /ostree/deploy/$OSNAME/$CHECKSUM
    - Each deployment is uniquely identified by SHA256 checksum
    - Each deployment has its own copy of /etc
- /usr is hard links to deploy directory
  - /usr is read-only
- Never boot to physical rootfs
  - initramfs chroot to “deployment”
- Persistent files can be stored in /var
- OSTree commands for deployment/rollback
  - ostree-admin-upgrade
  - ostree-admin-deploy {REFSPEC}
  - ostree-admin-status
  - ostree-admin-undeploy {INDEX}

Mount is established at boot time, pointing to the currently deployed filesystem

Containers

- Multiple container technologies available in Open Source
  - Docker
  - Flatpack
  - Resin
  - Snappy
- Container runtime — integrated into embedded device
  - Docker Engine — Most talked about
  - containerd — Simplified and robust container manager
  - runC — CLI based tool for starting and running containers
  - cri-o — container runtime for Kubernetes
- Ready for deployment applications have to be placed in containers before OTA update
- Physical location of app containers on target storage can be managed in similar way to regular application binary deployment

OTA process security

- SWUpdate example

Server authentication

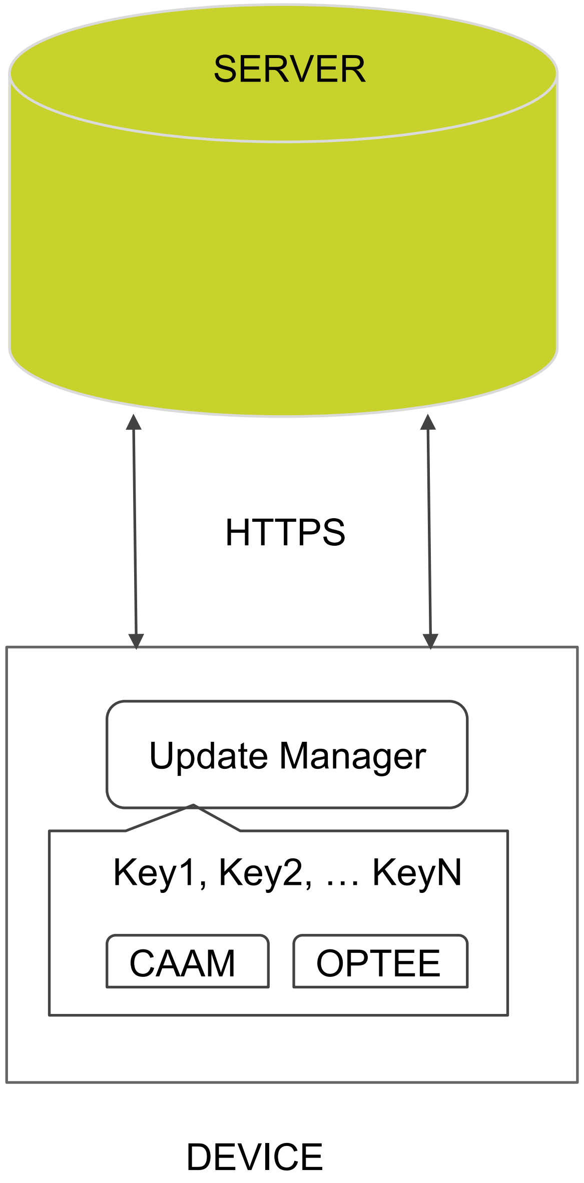
Why: Ensures device is downloading images from "trusted" server

How: PKI infrastructure with certificates (similar to how your browser trusts a website) using standard protocols such as HTTPS/TLS

Device: OTA Server's public key is stored on device to authenticate server.

OTA Server: OTA Server's private key is stored on server.

Optionally (highly recommended) signed certificates issued from a certificate authority is used to authenticate the server. This allows for easy migration of the update server between various cloud providers.



Device authentication

Why: Ensures “authorized/trusted” devices get updates and “counterfeit” devices or rouge actors not able to download firmware

How: Similar to OTA server authentication but the device public key is stored on the OTA server to authenticate the device.

The device private key can be stored securely on the device. **CAAM** blob in combination with **OP-TEE** can be used to protect the private key.

Note: The key generation is device specific and is typically done as part of the manufacturing protection mechanism process.

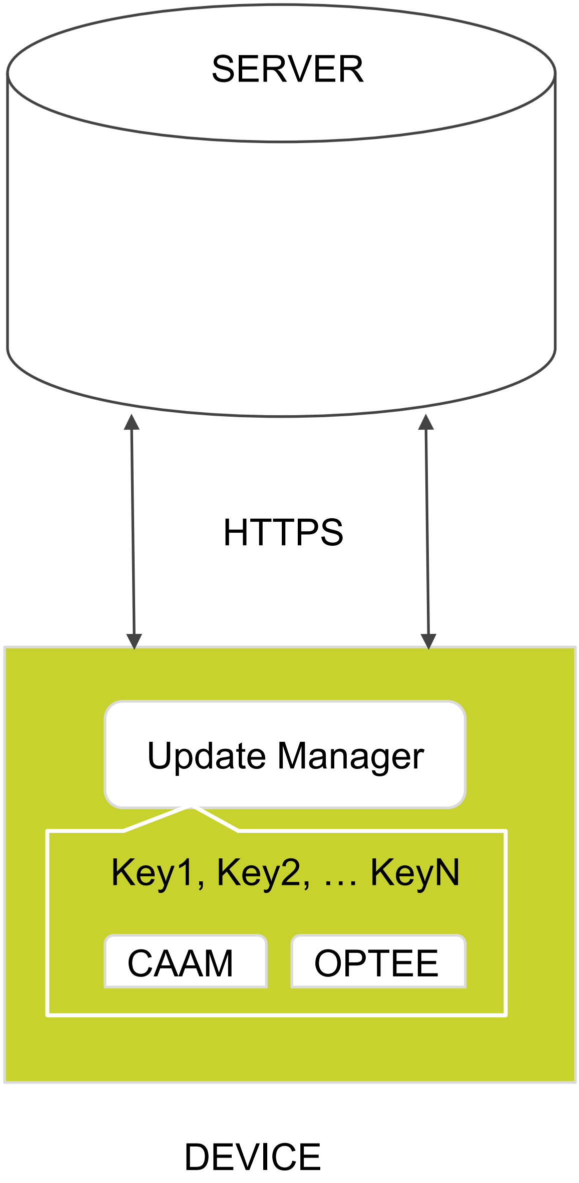


Image bundle authentication

Why: Ensures authenticity of image i.e. image is from the device manufacturer.

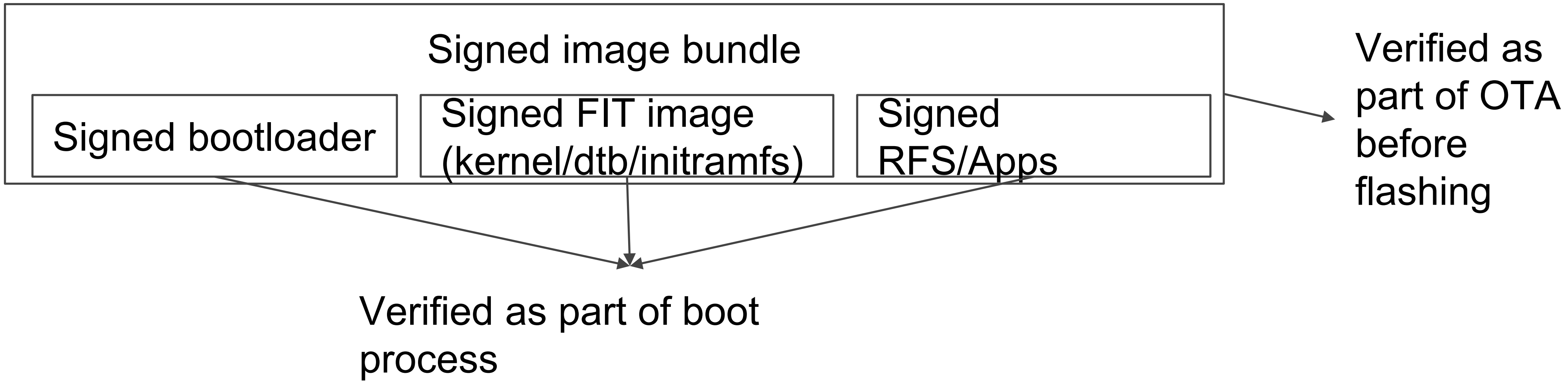
How: Image bundle (kernel/dtb/rfs/application) is signed using a private key on the build server or manufacturer signing server.

Public key is stored on device and used to authenticate the image **“before”** flashing.

Optionally the image can be encrypted using a shared symmetric key. The encryption key should be protected on the device using CAAM and/or OP-TEE.

Note: The keys mentioned above are different from the OTA server / device keys

Image bundle authentication (contd.)



Signed image bundle

Signed bootloader

Signed FIT image (kernel/dtb/initramfs)

Signed RFS/Apps

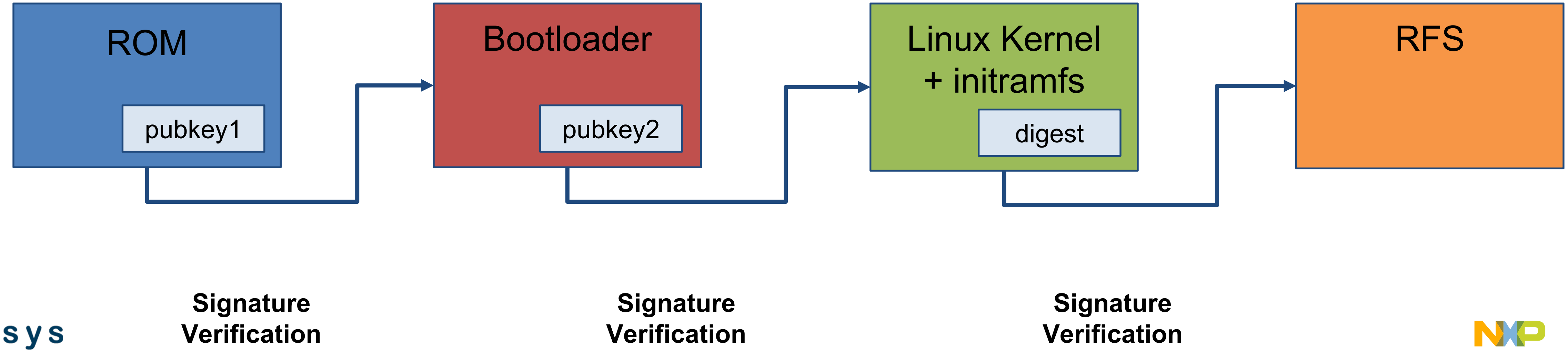
Verified as part of OTA before flashing

Verified as part of boot process

Individual component verification

Why: Protects against offline attacks (flash being replaced) or firmware being flashed using manufacturing tool etc.

How: This is part of the previously discussed secure boot and chain of trust where each component is signed individually using a private key on the build/signing server and the hash of the public key is stored on the i.MX processor OTP secure boot fuses to establish the hardware root of trust. The ROM code verifies the signed bootloader using the public key (verified from OTP flash) and bootloader verifies the kernel.. So on establishing the chain of trust.



Optional additional layer of security

In addition to storing secret keys in the CAAM, the i.MX Secure World OS OP-TEE can be used to authenticate and/or de-crypt the firmware update images to reduce the attack surface.

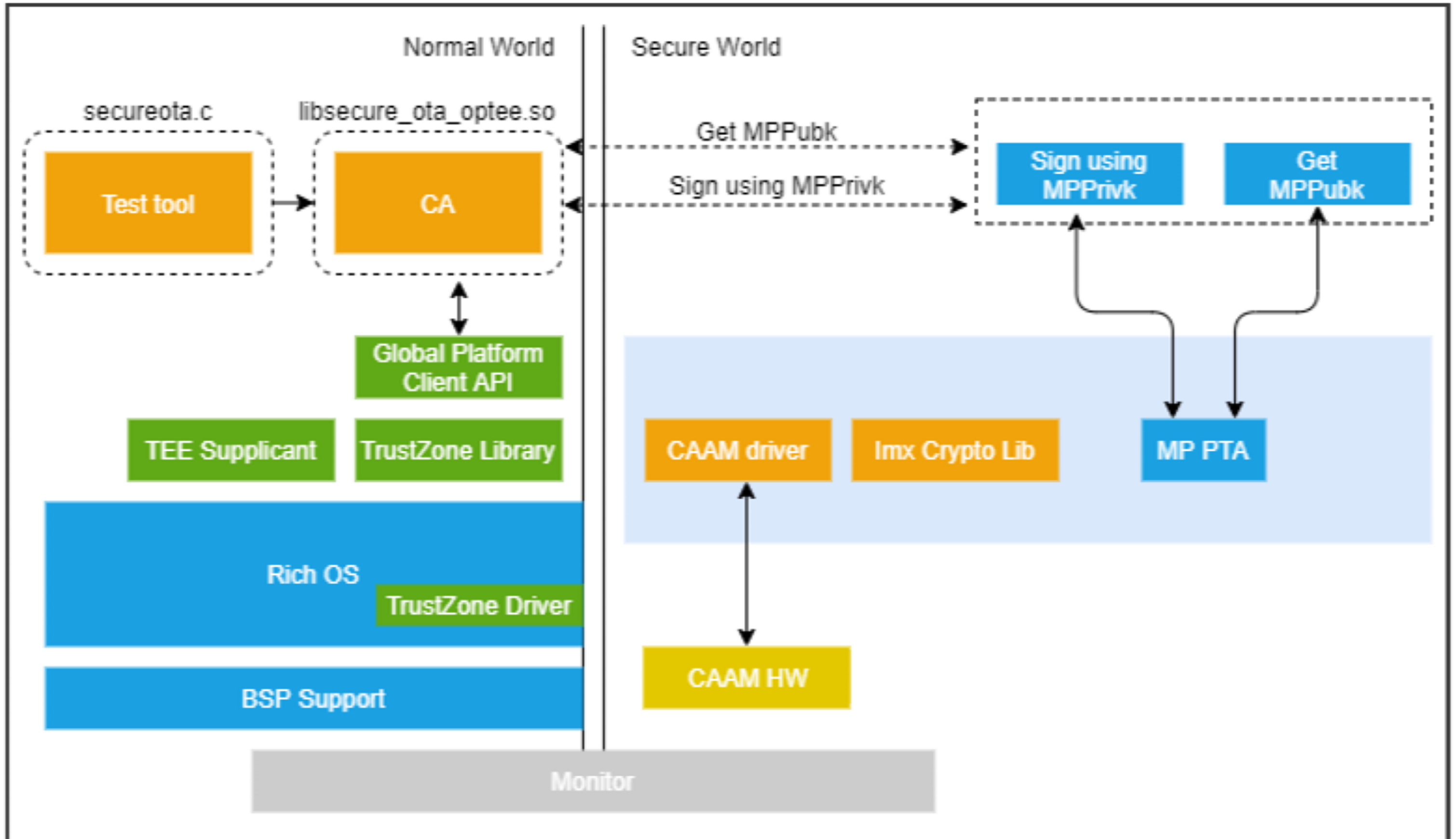
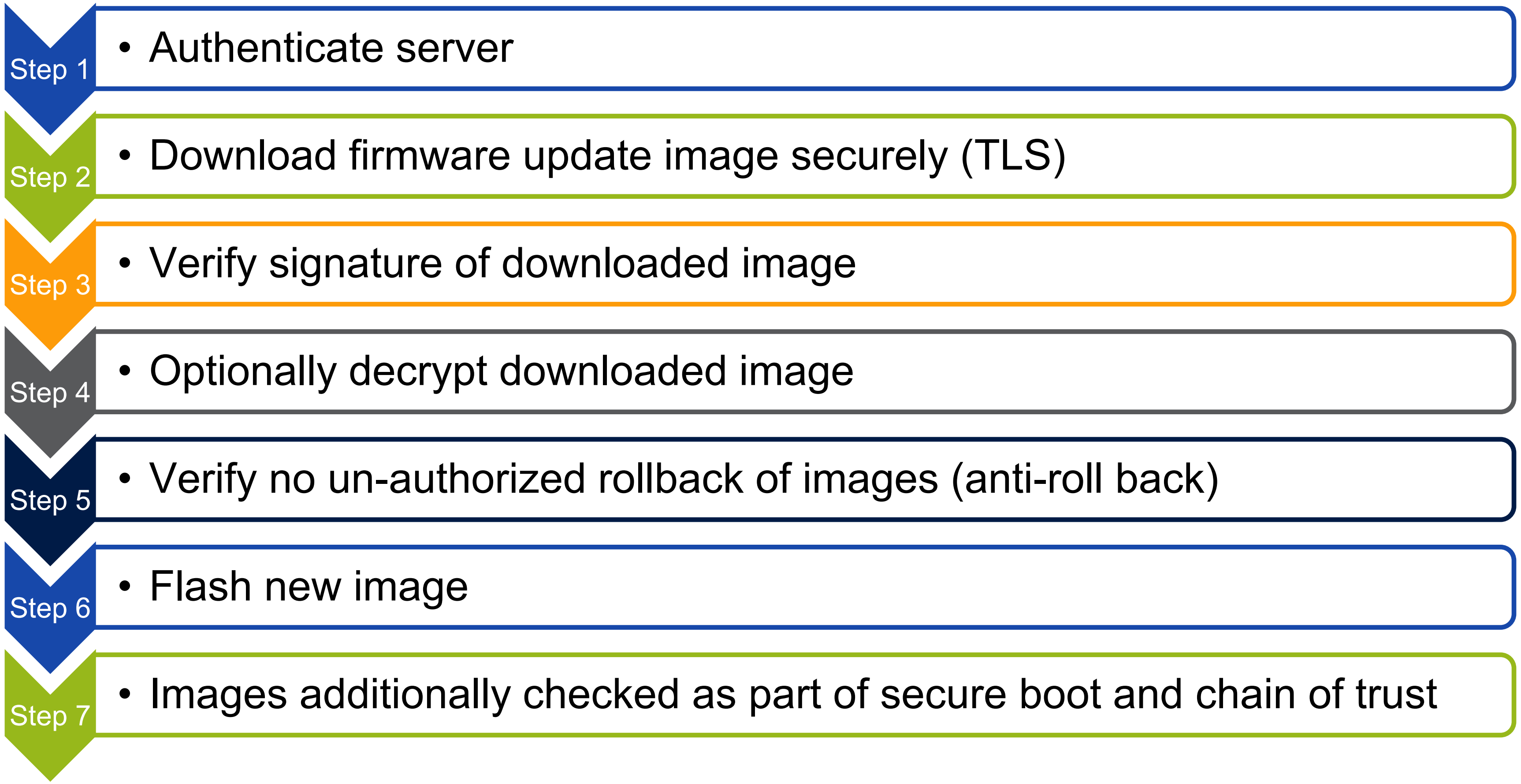


Figure 3. Overview of the OP-TEE application for MP

Source: NXP Application Note AN12900

OTA Security — rollout process

Actions undertaken typically by the update agent



Step 1
- Authenticate server

Step 2
- Download firmware update image securely (TLS)

Step 3
- Verify signature of downloaded image

Step 4
- Optionally decrypt downloaded image

Step 5
- Verify no un-authorized rollback of images (anti-roll back)

Step 6
- Flash new image

Step 7
- Images additionally checked as part of secure boot and chain of trust

Deployment management

Deployment of software updates requires:

- Information on available targets
    - Hardware revision
    - Currently installed software version
    - Device status (online/offline/available power/in-use)
- Information on available software updates
    - Software version
    - Intended hardware rev

Example solution of open source deployment framework – hawkBit

- Eclipse plugin

    https://eclipse.org/hawkbit

    - hawkBit is a domain-independent back-end framework for rolling out software updates to constrained edge devices as well as more powerful controllers and gateways connected to IP based networking infrastructure
    - Has integration with swupdate

Solution from Timesys

VigiShield Secure By Design Suite

- End-to-End Secure by Design solution for your product
- Utilizes full capabilities of underlying hardware and OS
- Best-in-class security features for your Linux platform
  - Software integrity and authentication
  - Data confidentiality (at rest, in use, in motion)
  - IP protection; anti-cloning
  - Reduces attack surface
  - Data isolation
- Integrates SWUpdate based OTA infrastructure

Confidentiality
Integrity
Data Security
Availability

Takeaways

- Field Updates capability is a must have requirement in products today
- Updates == ability to keep software in your products up to date
  - Ongoing Security
  - New features and bug fixes
- Many options and techniques are available (including Open Source)
- Need to consider system update backend when choosing the update technology
- Designing and implementing update system can be time consuming
- Accelerate design and implementation of OTA with VigiShield (for SWUpdate)
- Additional information on NXP security features:

  https://community.nxp.com/t5/i-MX-Security-Features-and/i-MX-Security-Features-amp-Collateral/ta-p/1192496

GOLD PARTNER NXP

Upcoming Webinars

NXP | SECURE CONNECTIONS FOR A SMARTER WORLD

In-depth Dive

- **Security Hardening:** Protecting Your Embedded Linux Device from the Risk of Being Compromised

Previous Webinars

Previous Webinars

Secure By Design Series

- Securing Embedded Linux Devices: Pitfalls to Avoid
- Software integrity and data confidentiality: Establishing secure boot and chain of trust on i.MX processors
- Trusted Execution Environment: Getting started with OP-TEE on i.MX processors
- Linux Kernel Security: Overview of Security Features and Hardening

Stay Secure (Vigiles) Series

- Software Security Management: Cutting through the vulnerability storm with NXP Vigiles
- BSP security maintenance: Best practices for vulnerability monitoring & remediation
- Full Life Cycle Security Maintenance of Embedded Linux BSPs
- Best practices for triaging Common Vulnerabilities & Exposures (CVEs) in embedded systems

For More Information and to Become More Secure

timesys

GOLD PARTNER NXP

Timesys is an embedded Linux security expert and NXP Gold Partner.
To discuss your project, please contact us at sales@timesys.com

Use this link to go to Services for securing your device

Thank You!

Q&A

SECURE CONNECTIONS
FOR A SMARTER WORLD