SCRATCh

SECURE AND AGILE CONNECTED THNGS

Deliverable

SCRATCh
nviso
sirris
driving industry by technology
consider it
Deutsches Forschungszentrum für Künstliche Intelligenz GmbH
Diebold Nixdorf

Deliverable 2.5: The Firmware Update System

Work package:	WP2
Affected milestone:	MS5
Partners involved:	Almende BV
	Diebold Nixdorf
	AnyWi
Date:	7/3/2022
Deliverable version:	v2.0
Editor:	Merijn van Tooren, Almende BV
Author(s):	Merijn van Tooren, Almende BV
	Peter Guenther, Diebold Nixdorf
	Marcell Marosvolgyi, AnyWi
Responsible Contact:	Merijn van Tooren
	Almende BV
	Stationsplein 45, unit a1.205-207, 3013 AK Rotterdam
	merijn@almende.org
	+316-10657644

Version history

Date	Version	Author	Comment
14/10/2020	1.0	Merijn van Tooren	
4/3/2022	2.0	Merijn van Tooren	

Table of Contents

1. Introduction

As a major part of the SCRATCh project, a new Firmware Update System is designed and developed. The goal is to create a new approach to setting up automatic, secure firmware updates for IoT end devices, and to create it so that the approach is user friendly, requires little to no security expertise, and is applicable to many if not all end devices.

As SCRATCh aims to help SMEs set up and perform Secure DevOps from scratch, it is important to investigate the matter of setting up a secure and effective firmware update system in any use case. Security by design is important here, as many interdependent components have to be set up on servers and end devices, and a large number of threats must be identified and addressed before the system may be considered secure.

This sub-project also involves working out how to plan secure communications and storage for many types of devices and data. Payloads of varying sizes must be hashed, signed and transported so they may be installed on end devices, and it may be necessary to store these updates for later use, yet conversely outdated updates form a security risk.

In this document, the process that lead to the Firmware Update System design is elaborated upon, and the final system draft is presented.

2. The Manifest

A first step in this undertaking was to study related works and create an overall strategy. The most relevant works studied are TUF (The Update Framework), Uptane, MCUBoot, Arm Platform Security Architecture, and the IETF Suit for Manifests. The essence drawn from these works is that there should be a "Manifest" containing metadata about an update, which is packaged and signed together with the update. Such a manifest contains data that will make it possible to ensure that an update is only deployed on intended target devices, and only at intended times.

What follows now is an analysis of security threats pertaining to firmware updates. These threats have been drawn from aforementioned literature research, and will be used to arrive at a list of requirements for the Firmware Update System's own Manifests. Threats that should be addressed outside the specific scope of the Firmware Update System are left out for the sake of conciseness. Threats are also categorised using STRIDE for convenience.

2.1. Threat Analysis

- **Image Tampering** (STRIDE Category: Tampering)
  - Important data / code is intercepted on its way to the end device and tampered with or replaced, then delivered as if normal.
- **Data tampering** (STRIDE Category: Tampering)
  - Important data / code is tampered with on device, e.g. while it is in flash memory.
- **Rollback** (STRIDE Category: Tampering)
  - An old update is provided to an end device to reintroduce old vulnerabilities.
- **Off-Chip Data Reading** (STRIDE Category: Information Disclosure)
  - Important data / code is read from the device, e.g. to create clones or for reverse engineering.
  - Not addressed using the Manifest. Rather, the device and off-chip storage must be designed in a way to prevent unwanted reading.
- **Persistent Malware** (STRIDE Category: Tampering, Elevation of Privilege)
  - Malicious software is installed on device in a way that persists across resets.
- **Update Abuse** (STRIDE Category: Denial of Service)
  - An attacker renders a device inoperable by sending an unmodified but incompatible firmware update, or by initiating many transfers without finishing them.
- **Side Channels** (STRIDE Category: Information Disclosure)

  - An attacker infers the value of sensitive on-chip code or data by using non-invasive techniques, such as differential power analysis or software observable side channels.
  - Note: Not addressed by the Firmware Update System, but considered an important threat to regard when setting up an update chain, and therefore included here.
- **Weak Cryptography** (STRIDE Category: Elevation of Privilege)
  - An attacker breaks the cryptography used by the end device.
  - Note: This is mainly included as a reminder that cryptography must be kept up to date in order to be fully effective.
- **Untrusted Installation** (STRIDE Category: Tampering, Denial of Service)
  - When a device awaits to receive an update, an attacker may provide untrusted files with any type or content, which may affect the device when installed.
- **Endless Data** (STRIDE Category: Denial of Service)
  - An attacker responds to a device with endless data, thereby hampering the device's system.
- **Extraneous Dependencies** (STRIDE Category: Tampering, Denial of Service)
  - A device's update system automatically downloads referenced dependencies with unwanted results.
- **Fast Forward** (STRIDE Category: Denial of Service)
  - An attacker increases meta data version number, e.g, to prevent subsequent updates due to downgrade protection.
- **Freeze** (STRIDE Category: Spoofing)
  - An attacker ensures that a device's requests for updates are answered with outdated firmware in order to render it unaware of the new updates it is being denied.
- **Mix and Match** (STRIDE Category: Tampering, Denial of Service)
  - Trick separate clients into installing a combination of firmware versions that did not exist together on server at the same time.
- **Key Compromise** (STRIDE Category: Elevation of Privilege)
  - An attacker obtains a complete set of required keys.
- **Read Updates** (STRIDE Category: Information Disclosure)
  - An attacker obtains sensitive data from update messages.
- **Deny Installation** (STRIDE Category: Denial of Service)
  - An attacker interferes with a device's communication and prevents it from receiving updates.
- **Interfere Functionality** (STRIDE Category: Tampering)

- An attacker causes a device to malfunction or exhibit unexpected behaviour, thereby causing harm.
- Note: This particular threat may result in bodily harm to persons on premises.

- **Control** (STRIDE Category: Elevation of Privilege)
  - An attacker gains full control of a device.
- **Mismatched Firmware** (STRIDE Category: Denial of Service)
  - An attacker sends a valid firmware image, for the wrong type of device, signed by an actor with firmware installation permission on both types of device. The firmware mismatch causes the device to fail.
- **Offline Device** (STRIDE Category: Tampering)
  - A device that has been without access to updates for a while is given an update that is new for it, yet outdated. Similar to [Rollback], this threat may result in introduction of old vulnerabilities.
- **Misinterpret Payload** (STRIDE Category: Denial of Service)
  - If a device misinterprets the type of the firmware image, it may cause a device to install a firmware image incorrectly.
- **Wrong Location** (STRIDE Category: Denial of Service)
  - If a device installs firmware to the wrong location, it may result in device failure.
- **Redirection** (STRIDE Category: Tampering, Denial of Service)
  - A device that attempts to download update data is redirected to a compromised server.
- **Verification Bypass** (STRIDE Category: Tampering)
  - An attacker replaces a newly downloaded firmware after a device finishes verifying a manifest.
- **Precursor Image** (STRIDE Category: Denial of Service)
  - An attacker sends a valid, current manifest to a device that has an unexpected precursor image.
- **Unqualified Image** (STRIDE Category: Elevation of Privilege)
  - Firmware that was not approved is introduced into deployment system.
- **Image Reverse Engineering** (STRIDE Category: Information Disclosure)
  - An attacker obtains a firmware image and reverse engineers it, e.g. to set up a tampering or to find vulnerabilities.
- **Manifest Override** (STRIDE Category: Elevation of Privilege)
  - An authorised actor, but not the firmware authority, uses an override mechanism to change an information element in a manifest signed by the firmware authority.

- **Extra Data** (STRIDE Category: Tampering)
    - Extra code is appended to a valid, authenticated image to smuggle unwanted software on device.
- **Manifest Modification** (STRIDE Category: Tampering)
    - The metadata manifest is modified after construction but before it is signed.
- **Brick Device** (STRIDE Category: Denial of Service)
    - An attacker sends crafted data to brick the device.

Following is a section in which requirements for the metadata Manifest are listed to address the above listed threats.

2.2. Manifest Requirements

- **Payload Digest**
    - A digest of the payload is included in the metadata.
    - Addresses Threats:
        - Image Tampering attacks are prevented by checking the payload with the digest in the metadata.
        - Data Tampering attacks are prevented by checking the payload with the digest in the metadata.
- **Metadata Signature**
    - The metadata must be signed so its authenticity can be verified. Included in metadata.
    - Addresses Threats:
        - The Image Tampering threat includes tampering with the image metadata, which is prevented by checking this signature.
- **Monotonic Sequence Numbers**
    - A version number that increases with every new release. Included in metadata.
    - Addresses Threats:
        - Rollback attacks are prevented by checking the monotonic sequence number in the metadata.
- **Vendor, Device-type Identifiers**
    - An identifier for the vendor and device type associated with a release. Included in metadata.
    - Addresses Threats:
        - Mismatched Firmware situations are prevented by checking the device-type identifier.
- **Expiration Time**

- A date, potentially date and time, at which the release will be considered outdated and unsafe. Included in metadata.
    - Addresses Threats:
        - The Offline Device threat is prevented by checking the expiration time, provided the device has a trusted source of time to check with.
        - The impact of the Deny Installation threat can be prevented by stopping devices from running on outdated firmware.
- **Authenticated Payload Type**
    - Indication of the payload type for unpacking and parsing. Included in metadata.
    - Addresses Threats:
        - The Misinterpret Payload threat is prevented by acting according to the payload type specified in the metadata.
- **Authenticated Storage Location**
    - Indication of the location on the device where the payload must be installed. Included in metadata.
    - Addresses Threats:
        - The Wrong Location threat is prevented by assuming the location specified in the metadata.
- **Authenticated Remote Resource Location**
    - Indication of the location of payload to be fetched remotely by the device. Included in metadata.
    - Addresses Threats:
        - Redirection problems are prevented by checking the remote location specified in the metadata.
- **Authenticated Precursor Images**
    - If an update requires a precursor image, a digest of said precursor image must be included in the metadata and used to verify the precursor image on device.
    - Addresses Threats:
        - The Precursor Image threat is prevented by checking the precusor image digest in the metadata with the image currently installed on device.
- **Rights Require Authenticity**
    - If there are multiple different rights and roles, then the individual rights must be sure from the Metadata Signature.
    - Addresses Threats:

        - The Unqualified Image threat is prevented by verifying that the signing entity has all required rights for the update being deployed.
- **Payload Encryption**
    - The payload must be encrypted.
    - Addresses Threats:
        - Image Reverse Engineering is counteracted by encryption of the payload, because the attacker will have to break the encryption to read the payload.
- **Access Control**
    - The device must be programmed to verify that the signing party has all the necessary rights. Relates to Rights Before Authenticity.
    - Addresses Threats:
        - The Unqualified Image threat is prevented by ensuring that a device knows how to verify the signing entity's rights.
- **Encrypted Manifests**
    - It must be possible to encrypt the metadata.
    - Addresses Threats:
        - The Read Updates threat is addressed by making metadata unreadable.
- **Whole Image Digest**
    - A digest of the complete installed payload. For fixed-storage devices, this should be a digest of the complete storage space, which allows verification against late unwanted additions.
    - Addresses Threats:
        - This counteracts the Extra Data threat by providing a way to check for extra data after installation.
- **Secure Reporting**
    - Reports and updates coming from the device should be authenticated to prevent spoofing or tampering.
    - Addresses Threats:
        - A form of the Read Updates threat is prevented by way of this requirement.
- **Protected Storage of Signing Keys**
    - Signing keys should be stored in such a way that they cannot be obtained by network access, and are unlikely to be usable from compromised development computers.
    - Addresses Threats:

        - The Key Compromise threat is addressed by this requirement, by making it harder for attackers to compromise any signing keys and use them.

- **Validate Manifests Prior to Deployment**
    - Metadata should be parsed and verified before deployment to verify they have not been tampered with.
    - Addresses Threats:
        - Addresses the threat of Manifest Modification.
- **Construct manifests in a trusted environment**
    - Metadata should be constructed in a trusted environment where tampering is unlikely.
    - Addresses Threats:
        - Addresses the threat of Manifest Modification.
- **Manifest kept immutable between check and use**
    - The metadata must be held immutable, and the end device must protect the metadata from other processes during the update process.
    - Addresses Threats:
        - Addresses a form of ImageTampering.
- **Immutable Boot Loader**
    - The immutable bootloader is a hardware Root of Trust that executes from reset, containing the minimal functionality required to check the authenticity of the Trusted Boot software.
    - Addresses Threats:
        - The Data Tampering threat is prevented by relying on the hardware root of trust provided by the immutable boot loader.
- **Authenticated Trusted Boot Stages**
    - Following the Immutable Boot Loader, each next stage of boot must be verified by the previous.
    - Addresses Threats:
        - The Data Tampering threat is prevented by ensuring that all boot stages are, by extension, verified by the immutable boot loader.
- **Chain of Trust**
    - A chain of trust ensures that each loaded component on the system has not been tampered. The chain of trust begins on reset whereby a hardware component authenticates the first stage of software. The chain continues when the authenticated software loads additional software.

- Addresses Threats:
  - Counteracts Persistent Malware.

– **Identified Roles**
  - A product analysis must determine the number of potential actors that will maintain the components of a product.
  - Addresses Threats:
    - Rather than directly addressing a threat, this requirement is necessary for the proper implementation of requirements related to having multiple signing parties.

– **Signer ID**
  - Identifies the signing party. Included in metadata.
  - Addresses Threats:
    - This requirement enables other requirements concerned with verifying that a signing party has the necessary rights to deploy an update.

– **Manifest Format Version**
  - The metadata format version. Included in metadata.
  - Addresses Threats:
    - A form of Update Abuse is prevented by avoiding metadata format version mismatch.

– **Payload Size**
  - The size of the payload. Included in metadata.
  - Addresses Threats:
    - The Endless Data threat is prevented by including a payload size and reading no more than indicated.
    - The Extra Data threat is also counteracted by reading none more of the payload than indicated.

– **Role Indicator**
  - If there are multiple signing roles, the role of the signing party must be indicated in the metadata.
  - Addresses Threats:
    - This requirement keeps other requirements consistent in the presence of multiple roles, e.g. in an Uptane implementation.

– **Root Metadata**
  - The Root Metadata identifies the roles, their public keys, and the threshold of signatures required for those roles.
  - Addresses Threats:
    - This requirement keeps other requirements consistent in the presence of multiple roles, e.g. in an Uptane implementation.

– **Snapshot Metadata**

- The Snapshot Metadata lists the version numbers and file names of Targets Metadata files.
- Addresses Threats:
  - This requirement prevents a specific form of attack where Targets Metadata files are mixed-and-matched to create an invalid combination of valid file and version number.
- Research Note:
  - The Uptane Standard has a very specific set of four roles, namely Root, Targets, Snapshot and Timestamp, and some of its requirements inevitably refer to this setup. Refer to the original document for more information.

– **Timestamp Metadata**
- The Timestamp Metadata contains the version number and file name of the latest Snapshot Metadata file, as well as at least one digest of that file.
- Addresses Threats:
  - This requirement makes it possible to verify whether an update is the latest update. It can therefore be used to counter the Freeze threat.

– **End-Device Requirements**
- An Uptane-compliant ECU shall be able to download and verify Image metadata and image binaries before installing a new image and MUST have a secure way of verifying the current time, or a sufficiently recent attestation of the time.
- Addresses Threats:
  - A consequence of other requirements, as such devices must be able to run security code before installing downloaded payload, and must check whether the payload is outdated.

3. System Architecture

Before getting into the design proper, this document will elaborate upon the discussion of push and pull architecture as an interesting aspect of the research.

3.1. Push and Pull

In order for updates to be applied automatically, there must be an automatic initiative. Typically, this is push or pull: either the owner of the update pushes it towards devices, or the devices pull new updates from a repository. This is a choice with consequences, and there are arguments against both approaches.

Firstly, for a push approach, it must be possible to send a message to an end device in order to get it to accept an update. If an end device is listening for such a push, that is also a vulnerability, as an attacker will also be capable of triggering or even providing updates through this channel. Secondly, this requires that the end device is kept alive and listening, and may introduce an unwanted drain of batteries or other resource. It is also very significant that, in order to be able to push to an end device, that end device must be known. If an end device is not known to the device manager, it will not receive updates, and if it is tampered with, its changed behaviour will not be noticed during update attempts.

Conversely, in a pull approach, the initiative is the responsibility of the end devices. This is a risk in and of itself. Typically, the end devices are much less accessible and debuggable remotely than the device manager - which is the reason to have this sort of system - and if an end device stops pulling updates for any reason, this is harder to detect as well as to fix. Furthermore, this approach takes a measure of control away from the device manager, which may have implications for security and for safety. If an update must be rushed or delayed, in a pull setup, the device manager can do nothing more than refuse pull attempts. There is no way to spontaneously trigger an update, nor to change the pull schedule without updating the end device.

There is somewhat of a third method, which is to set up a messaging service that allows end devices to "subscribe" to update notifications. In terms of security concerns, this is largely similar to a push setup, but it generally makes it easier for attackers to perform a fake push.

Based on these considerations, the conclusion is drawn that push and pull are optimal in different situations, depending on a weighing of security and safety threats, and the required control over the update schedule. Therefore, the update system should offer support for both approaches, and the user should be able to make a selection.

3.2. The Design

To reiterate, the goal is to design a system for firmware updates of multiple end devices. The pain point is a lack of uniformity – the end devices may be from different vendors, may receive updates via different channels, etcetera. The system must use a metadata Manifest to secure the update chain from attacks.

The design is thus: an “Update Manager” must be set up on site, and connected to end devices. It may be directly connected, or additional devices may be used as bridges. This amount of centralisation is accepted in order to create an overview of live firmware versions, which is considered an essential form of monitoring that will help counter many threats.

In an analysis of the specific use case, security requirements from the Manifest Requirements subsection will be selected to decide on the composition of the metadata Manifest. Firmware updates will be supplied with such Manifests, signed, and made available on an online repository.

The Update Manager will download firmware updates from the repository and act as a midway point of trust. It should be properly provisioned for this purpose. Provided that a downloaded update is verified by the Update Manager, it may be applied to the applicable end devices, as identified by the metadata.

For each type of end device, an Update Driver must be implemented. This should be a simple executable or process that can receive data through an internal socket within the Update Manager, and proceed to provide the update to connected, compatible end devices.

The Update Driver should use the same socket to report back to the Update Manager about the success or failure of updating each individual connected end device, and this information should be stored by the Update Manager. Appropriate staff should be notified of any failed updates. Further notifications should follow if end devices continue to go without (successful) updates for a certain amount of time.

4. Conclusion

After a lengthy research of the many threats involved in secure firmware updating, and the possible ways to counter them using a firmware update system with metadata manifests, a design for a generic solution has been achieved. The nature of the field means that no single full implementation will suffice for most use cases that exist, but the information gathered in this document should prove instrumental in tackling this aspect of a project's Secure DevOps.