

SUIT
Internet-Draft
Intended status: Informational
Expires: August 2, 2018

B. Moran
M. Meriac
H. Tschofenig
Arm Limited
January 29, 2018

A Firmware Update Architecture for Internet of Things Devices
draft-moran-suit-architecture-01

Abstract

Vulnerabilities with Internet of Things (IoT) devices have raised the need for a solid and secure firmware update mechanism that is also suitable for constrained devices. Incorporating such update mechanism to fix vulnerabilities, to update configuration settings as well as adding new functionality is recommended by security experts.

This document lists requirements and describes an architecture for a firmware update mechanism suitable for IoT devices. The architecture is agnostic to the transport of the firmware images and associated meta-data.

This version of the document assumes asymmetric cryptography and a public key infrastructure. Future versions may also describe a symmetric key approach for very constrained devices.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of [BCP 78](#) and [BCP 79](#).

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <http://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on August 2, 2018.

Copyright Notice

Copyright (c) 2018 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to [BCP 78](http://trustee.ietf.org/license-info) and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

This document may contain material from IETF Documents or IETF Contributions published or made publicly available before November 10, 2008. The person(s) controlling the copyright in some of this material may not have granted the IETF Trust the right to allow modifications of such material outside the IETF Standards Process. Without obtaining an adequate license from the person(s) controlling the copyright in such materials, this document may not be modified outside the IETF Standards Process, and derivative works of it may not be created outside the IETF Standards Process, except to format it for publication as an RFC or to translate it into languages other than English.

Table of Contents

1. Introduction	4
2. Conventions and Terminology	4
3. Requirements	5
3.1. Agnostic to how firmware images are distributed	5
3.2. Friendly to broadcast delivery	6
3.3. Uses state-of-the-art security mechanisms	6
3.4. Rollback attacks must be prevented.	6
3.5. High reliability	6
3.6. Minimal bootloader	6
3.7. Minimal impact on existing firmware formats	7
3.8. Robust permissions	7
3.8.1. Policy documents	7
4. Architecture	8
5. Manifest	9
6. Example Flow	10
7. IANA Considerations	12
8. Security Considerations	12
9. Mailing List Information	13
10. Acknowledgements	13

11. Appendix A: Threat Model, User Stories, Security Requirements, and Usability Requirements	14
11.1. Threat Model	14
11.2. Threat Descriptions	14
11.2.1. Threat MFT1: Old Firmware	14
11.2.2. Threat MFT2: Mismatched Firmware	14
11.2.3. Threat MFT3: Offline device + Old Firmware	15
11.2.4. Threat MFT4: The target device misinterprets the type of payload	15
11.2.5. Threat MFT5: The target device installs the payload to the wrong location	15
11.2.6. Threat MFT6: Redirection	16
11.2.7. Threat MFT7: Payload Verification on Boot	16
11.2.8. Threat MFT8: Unauthenticated Updates	16
11.2.9. Threat MFT9: Unexpected Precursor images	16
11.2.10. Threat MFT10: Unqualified Firmware	16
11.3. Security Requirements	17
11.3.1. Security Requirement MFSR1: Monotonic Sequence Numbers	17
11.3.2. Security Requirement MFSR2: Vendor, Device-type Identifiers	18
11.3.3. Security Requirement MFSR3: Best-Before Timestamps	18
11.3.4. Security Requirement MFSR4: Signed Payload Descriptor	18
11.3.5. Security Requirement MFSR5: Cryptographic Authenticity	19
11.3.6. Security Requirement MFSR6: Rights Require Authenticity	19
11.4. User Stories	19
11.4.1. Use Case MFUC1: Installation Instructions	19
11.4.2. Use Case MFUC2: Reuse Local Infrastructure	20
11.4.3. Use Case MFUC3: Modular Update	20
11.4.4. Use Case MFUC4: Multiple Authorisations	20
11.4.5. Use Case MFUC5: Multiple Payload Formats	20
11.5. Usability Requirements	20
11.5.1. Usability Requirement MFUR1:	21
11.5.2. Usability Requirement MFUR2:	21
11.5.3. Usability Requirement MFUR3:	21
11.5.4. Usability Requirement MFUR4:	21
11.5.5. Usability Requirement MFUR5:	21
11.6. Manifest Fields	22
11.6.1. Manifest Field: Timestamp	22
11.6.2. Manifest Field: Vendor ID Condition	22
11.6.3. Manifest Field: Class ID Condition	22
11.6.4. Manifest Field: Precursor Image Digest Condition	22
11.6.5. Manifest Field: Best-Before timestamp	23
11.6.6. Manifest Field: Payload Format	23
11.6.7. Manifest Field: Storage Location	23

11.6.8.	Manifest Field: URIs	23
11.6.9.	Manifest Field: Digests	23
11.6.10.	Manifest Field: Size	23
11.6.11.	Manifest Field: Signature	24
11.6.12.	Manifest Field: Directives	24
11.6.13.	Manifest Field: Aliases	24
11.6.14.	Manifest Field: Dependencies	24
12.	References	24
12.1.	Normative References	24
12.2.	Informative References	25
12.3.	URIs	25
	Authors' Addresses	25

1. Introduction

When developing IoT devices, one of the most difficult problems to solve is how to update the firmware on the device. Once the device is deployed, firmware updates play a critical part in its lifetime, particularly when devices have a long lifetime, are deployed in remote or inaccessible areas or where manual intervention is cost prohibitive or otherwise difficult. The need for a firmware update may be to fix bugs in software, to add new functionality, or to re-configure the device.

The firmware update process has to ensure that

- The firmware image is authenticated and attempts to flash a malicious firmware image are prevented.
- The firmware image can be confidentiality protected so that attempts by an adversary to recover the plaintext binary can be prevented. Obtaining the plaintext binary is often one of the first steps for an attack to mount an attack.

2. Conventions and Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC 2119](#) [RFC2119].

This document uses the following terms:

- Manifest: The manifest contains meta-data about the firmware image. The manifest is protected against modification and provides information about the author.

- **Firmware Image:** The firmware image is a binary that may contain the complete software of a device or a subset of it. The firmware image may consist of multiple images, if the device contains more than one microcontroller. The image may consist of a differential update for performance reasons.

The following entities are used:

- **Author:** The author is the entity that creates the firmware image, signs and/or encrypts it and attaches a manifest to it. The author is most likely a developer using a set of tools.
- **Device:** The device is the recipient of the firmware image and the manifest. The goal is to update the firmware of the device.
- **Untrusted Storage:** Firmware images and manifests are stored on untrusted filesystems or cloud storage infrastructure. Some deployments may require storage of the firmware images/manifests to be stored on various entities before they reach the device.

3. Requirements

The firmware update mechanism described in this specification was designed with the following requirements in mind:

- Agnostic to how firmware images are distributed
- Friendly to broadcast delivery
- Uses state-of-the-art security mechanisms
- Operates with a small bootloader
- Minimal impact on existing firmware formats
- Robust permissions

3.1. Agnostic to how firmware images are distributed

Firmware images can be conveyed to devices in a variety of ways, including USB, UART, WiFi, BLE, low-power WAN technologies, etc and use different protocols (e.g., CoAP, HTTP). The specified mechanism needs to be agnostic to the distribution of the firmware images and manifests.

3.2. Friendly to broadcast delivery

For an update to be broadcast friendly, it cannot rely on link layer, network layer, or transport layer security. In addition, the same message must be deliverable to many devices; both those to which it applies and those to which it does not without a chance that the wrong device will accept the update. Considerations that apply to network broadcasts apply equally to the use of third-party content distribution networks for payload distribution.

3.3. Uses state-of-the-art security mechanisms

End-to-end security between the author and the device, as shown in [Section 4](#), is used to ensure that the device can verify firmware images and manifests produced by authorized authors.

If the firmware image is to be encrypted, it must be done in such a way that every intended recipient can decrypt it. The information that is encrypted individually for each device must be an absolute minimum.

3.4. Rollback attacks must be prevented.

All information necessary for a device to make a decision about the installation of a firmware update must fit into the available RAM of a constrained IoT device. This prevents flash write exhaustion.

Since parsers are known sources of bugs it must be easy to parse only those fields which are required to validate at least one signature with minimal exposure.

3.5. High reliability

A power failure at any time must not cause a failure of the device. A failure to validate any part of an update must not cause a failure of the device. To achieve this, the device is required to provide a minimum of two storage locations for firmware and one bootable location for firmware. Note: This is an implementation requirement rather than a requirement on the manifest format.

3.6. Minimal bootloader

The bootloader must be minimal, containing only the flash and cryptographic primitives necessary to read the stored firmware, validate the received firmware, and write the bootable firmware. The bootloader should not require updating, since a failed update poses a risk in reliability. If more functionality is required in the bootloader, it must use a two-stage bootloader, with the first stage

comprising the functionality defined above. Note: This is an implementation requirement.

3.7. Minimal impact on existing firmware formats

The design of the firmware update mechanism must not require changes to existing firmware formats.

3.8. Robust permissions

A device may have many modules that require updating individually. It may also need to trust several actors in order to authorize an update. For example, a firmware author may not have the authority to install firmware on a device in critical infrastructure without the authorization of a device operator. In this case, the device should reject firmware updates unless they are signed both by the firmware author and by the device operator. To facilitate complex use-cases such as this, updates require several permissions.

3.8.1. Policy documents

When a simple set of permissions fails to encapsulate the rules required for a device make decisions about firmware, a policy document can be used instead. Several policy documents can be used together, when multiple actors should have the rights to set policies.

Some example policies:

- Trust the actor identified by the referenced public key.
- Three actors are trusted identified by their public keys. Signatures from at least two of these actors are required to trust a manifest.
- The actor identified by the referenced public key is authorized to create secondary policies

The baseline policies for all manifests are described in [Appendix A](#). In summary, they are:

- Do not install firmware with earlier metadata than the current metadata.
- Only install firmware with a matching vendor, model, hardware revision, software version, etc.
- Only install firmware that is before its best-before timestamp.

- Only install firmware with metadata signed by a trusted actor.
- Only allow an actor to exercise rights on the device via a manifest if that actor has signed the manifest.
- Only allow a firmware installation if all required rights have been met through signatures (one or more) or manifest dependencies (one or more).
- Use the instructions provided by the manifest to install the firmware.
- Any authorized actor may redirect any URI.
- Install any and all firmware images that are linked together with manifest dependencies.
- Choose the mechanism to install the firmware, based on the type of firmware it is.

4. Architecture

The architecture graphically shown in Figure 1 illustrates that an author creates a firmware image and the corresponding manifest. The firmware image may be encrypted and will be integrity protected. The meta-data is integrity protected. When the author is ready to distribute the firmware image it conveys it using his or her favorite communication channel to the device, which will typically involve the use of untrusted storage, like a file server. Whether the firmware image and the manifest is pushed to the device or fetched by the device is outside the scope of this work and existing device management protocols can be used for efficiently distributing this information.

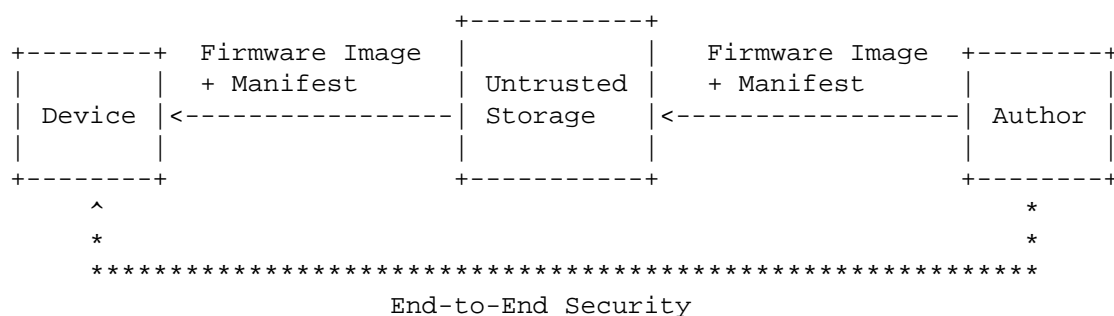


Figure 1: End-to-End Security Architecture.

The following assumptions are made to allow the device to verify the received firmware image and manifest before updating software:

- To accept an update, a device needs to decide whether the author signing the firmware image and the manifest is authorized to make the updates. We use public key cryptography to accomplish this. The device verifies the signature covering the manifest using a digital signature algorithm. The device is provisioned with a trust anchor that is used to validate the digital signature produced by the author. This trust anchor is potentially different from the trust anchor used to validate the digital signature produced for other protocols (such as device management protocols). This trust anchor may be provisioned to the device during manufacturing or during commissioning.
- For confidentiality protection of firmware images the author needs to be in possession of the certificate/public key of a device.

5. Manifest

In order for a device to apply an update, it has to make several decisions about the update:

- Does it trust the author of the update?
- Has the firmware been corrupted?
- Does the firmware update apply to this device?
- Is the update older than the active firmware?
- When should the device apply the update?
- How should the device apply the update?
- What kind of firmware binary is it?
- Where should the update be obtained?
- Where should the firmware be stored?

The manifest encodes the information that devices need in order to make these decisions. It is a data structure that contains the following information:

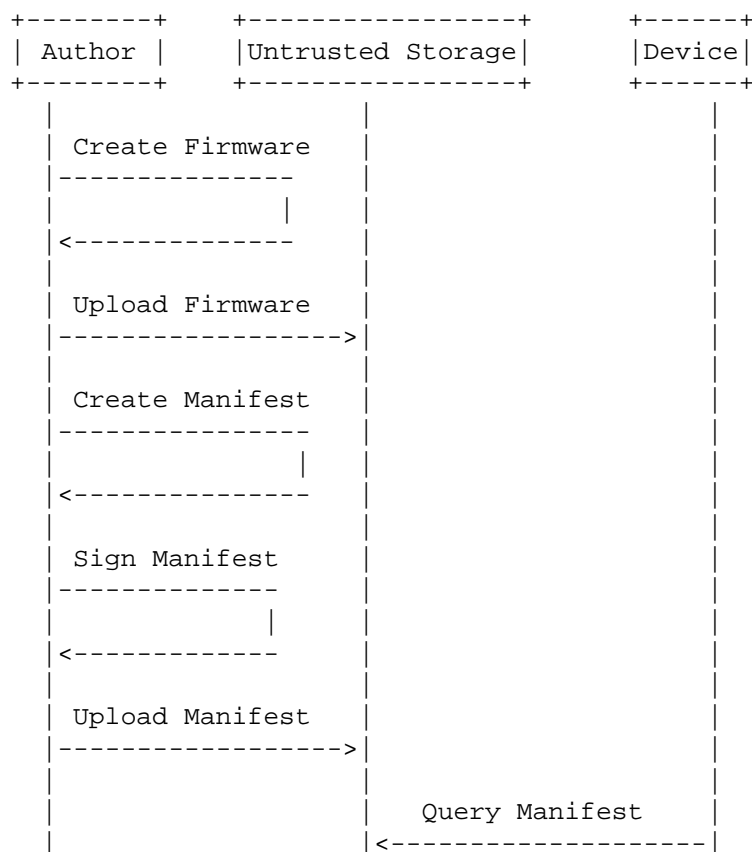
- information about the device(s) the firmware image is intended to be applied to,

- information about when the firmware update has to be applied,
- information about when the manifest was created,
- dependencies to other manifests,
- pointers to the firmware image and information about the format,
- information about where to store the firmware image,
- cryptographic information, such as digital signatures.

The manifest format is described in a companion document.

6. Example Flow

The following example message flow illustrates the interaction for distributing a firmware image to a device starting with an author uploading the new firmware to untrusted storage and creating a manifest.



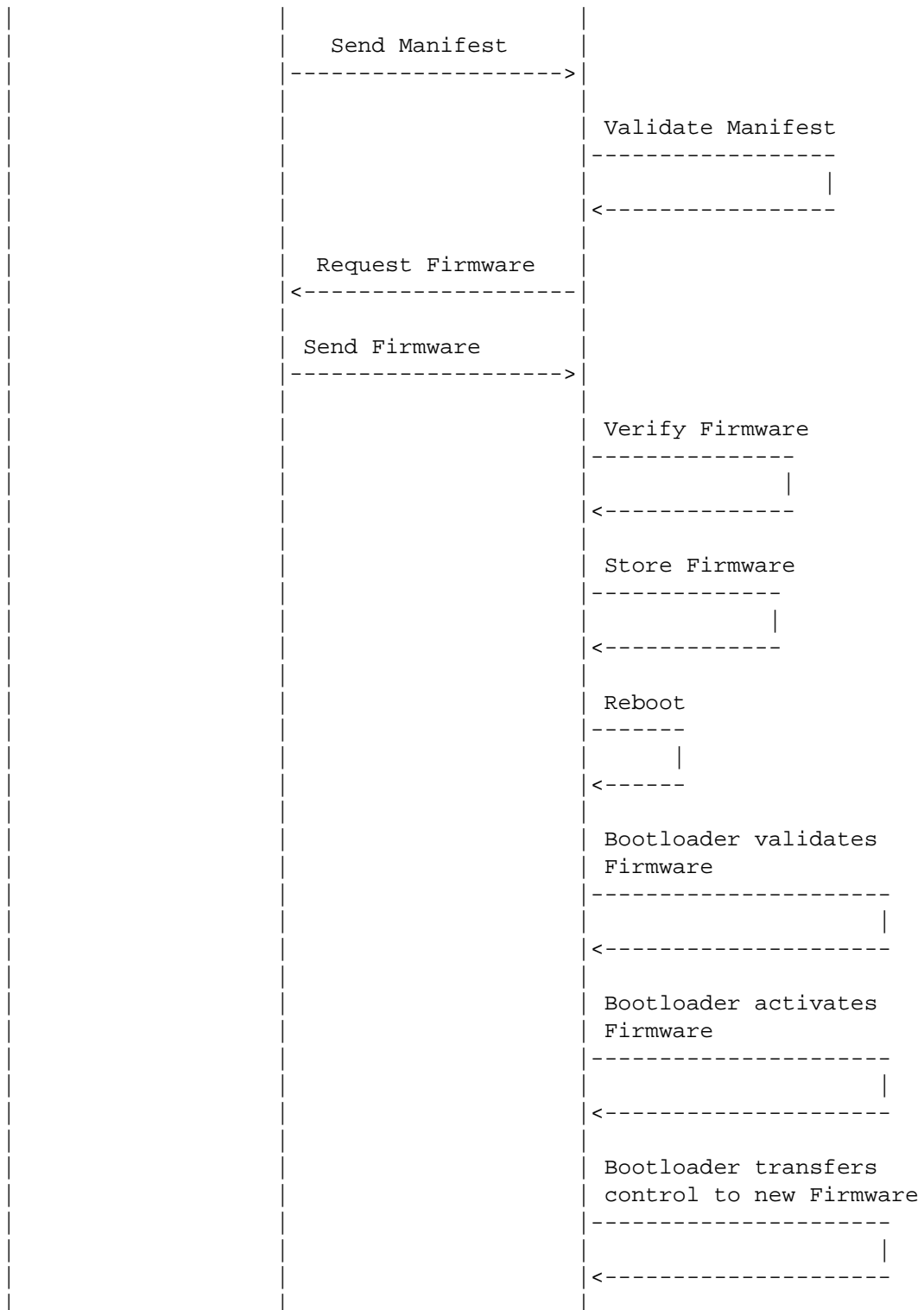


Figure 2: Example Flow for a Firmware Upate.

7. IANA Considerations

This document does not require any actions by IANA.

8. Security Considerations

Firmware updates fix security vulnerabilities and are considered to be an important building block in securing IoT devices. Due to the importance of firmware updates for IoT devices the Internet Architecture Board (IAB) organized a 'Workshop on Internet of Things (IoT) Software Update (IOTSU)', which took place at Trinity College Dublin, Ireland on the 13th and 14th of June, 2016 to take a look at the big picture. A report about this workshop can be found at [\[RFC8240\]](#). This document (and associated specifications) offer a standardized firmware manifest format providing end-to-end security from the author to the device.

There are, however, many other considerations raised during the workshop. Many of them are outside the scope of standardization organizations since they fall into the realm of product engineering, regulatory frameworks, and business models. The following considerations are outside the scope of this document, namely

- installing firmware updates in a robust fashion so that the update does not break the device functionality of the environment this device operates in.
- installing firmware updates in a timely fashion considering the complexity of the decision making process of updating devices, potential re-certification requirements, and the need for user's consent to install updates.
- the distribution of the actual firmware update, potentially in an efficient manner to a large number of devices without human involvement.
- energy efficiency and battery lifetime considerations.
- key management required for verifying the digital signature protecting the manifest.
- incentives for manufacturers to offer a firmware update mechanism as part of their IoT products.

9. Mailing List Information

The discussion list for this document is located at the e-mail address suit@ietf.org [1]. Information on the group and information on how to subscribe to the list is at <https://www1.ietf.org/mailman/listinfo/suit>

Archives of the list can be found at: <https://www.ietf.org/mail-archive/web/suit/current/index.html>

10. Acknowledgements

We would like the following persons for their feedback:

- Geraint Luff
- Amyas Phillips
- Dan Ros
- Thomas Eichinger
- Michael Richardson
- Emmanuel Baccelli
- Ned Smith
- David Brown
- Jim Schaad
- Carsten Bormann
- Cullen Jennings
- Olaf Bergmann
- Suhas Nandakumar
- Phillip Hallam-Baker

We would also like to thank the WG chairs, Russ Housley, David Waltermire, Dave Thaler and the responsible security area director, Kathleen Moriarty, for their support.

11. [Appendix A](#): Threat Model, User Stories, Security Requirements, and Usability Requirements

11.1. Threat Model

This appendix aims to provide information about the threats that were considered, the security requirements that are derived from those threats and the fields that permit implementation of the security requirements. This model uses the S.T.R.I.D.E. [[STRIDE](#)] approach. Each threat is classified according to:

- Spoofing Identity
- Tampering with data
- Repudiation
- Information disclosure
- Denial of service
- Elevation of privilege

11.2. Threat Descriptions

11.2.1. Threat MFT1: Old Firmware

Classification: Escalation of Privilege

An attacker sends an old, but valid manifest with an old, but valid firmware image to a device. If there is a known vulnerability in the provided firmware image, this may allow an attacker to exploit the vulnerability and gain control of the device.

Threat Escalation: If the attacker is able to exploit the known vulnerability, then this threat can be escalated to ALL TYPES.

11.2.2. Threat MFT2: Mismatched Firmware

Classification: 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 is verified by the device positively because it is signed by an actor with the appropriate permission. This could have wide-ranging consequences. For devices that are similar, it could cause minor breakage, or expose security

vulnerabilities. For devices that are very different, it is likely to render devices inoperable.

11.2.3. Threat MFT3: Offline device + Old Firmware

Classification: Escalation of Privilege

An attacker targets a device that has been offline for a long time and runs an old firmware version. The attacker sends an old, but valid manifest to a device with an old, but valid firmware image. The attacker-provided firmware is newer than the installed one but older than the most recently available firmware. If there is a known vulnerability in the provided firmware image then this may allow an attacker to gain control of a device. Because the device has been offline for a long time, it is unaware of any new updates. As such it will treat the old manifest as the most current.

Threat Escalation: If the attacker is able to exploit the known vulnerability, then this threat can be escalated to ALL TYPES.

11.2.4. Threat MFT4: The target device misinterprets the type of payload

Classification: Denial of Service

If a device misinterprets the type of the firmware image, it may cause a device to install a firmware image incorrectly. An incorrectly installed firmware image would likely cause the device to stop functioning.

Threat Escalation: An attacker that can cause a device to misinterpret the received firmware image may gain escalation of privilege and potentially expand this to all types of threat.

11.2.5. Threat MFT5: The target device installs the payload to the wrong location

Classification: Denial of Service

If a device installs a firmware image to the wrong location on the device, then it is likely to break. For example, a firmware image installed as an application could cause a device and/or an application to stop functioning.

Threat Escalation: An attacker that can cause a device to misinterpret the received code may gain escalation of privilege and potentially expand this to all types of threat.

11.2.6. Threat MFT6: Redirection

Classification: Denial of Service

If a device does not know where to obtain the payload for an update, it may be redirected to an attacker's server. This would allow an attacker to provide broken payloads to devices.

11.2.7. Threat MFT7: Payload Verification on Boot

Classification: All Types

An attacker replaces a newly downloaded firmware after a device finishes verifying a manifest. This could cause the device to execute the attacker's code. This attack likely requires physical access to the device. However, it is possible that this attack is carried out in combination with another threat that allows remote execution.

11.2.8. Threat MFT8: Unauthenticated Updates

Classification: All Types

If an attacker can install their firmware on a device, by manipulating either payload or metadata, then they have complete control of the device.

11.2.9. Threat MFT9: Unexpected Precursor images

Classification: Denial of Service

An attacker sends a valid, current manifest to a device that has an unexpected precursor image. If a payload format requires a precursor image (for example, delta updates) and that precursor image is not available on the target device, it could cause the update to break.

Threat Escalation: An attacker that can cause a device to install a payload against the wrong precursor image could gain escalation of privilege and potentially expand this to all types of threat.

11.2.10. Threat MFT10: Unqualified Firmware

Classification: Denial of Service, Escalation of Privilege

This threat can appear in several ways, however it is ultimately about interoperability of devices with other systems. The owner or operator of a network needs to approve firmware for their network in order to ensure interoperability with other devices on the network,

or the network itself. If the firmware is not qualified, it may not work. Therefore, if a device installs firmware without the approval of the network owner or operator, this is a threat to devices and the network.

Example 1: We assume that OEMs expect the rights to create firmware, but that Operators expect the rights to qualify firmware as fit-for-purpose on their networks.

An attacker obtains a manifest for a device on Network A. They send that manifest to a device on Network B. Because Network A and Network B are different, and the firmware has not been qualified for Network B, the target device is disabled by this unqualified, but signed firmware.

This is a denial of service because it can render devices inoperable. This is an escalation of privilege because it allows the attacker to make installation decisions that should be made by the Operator.

Example 2: Multiple devices that interoperate are used on the same network. Some devices are manufactured by OEM A and other devices by OEM B. These devices communicate with each other. A new firmware is released by OEM A that breaks compatibility with OEM B devices. An attacker sends the new firmware to the OEM A devices without approval of the network operator. This breaks the behaviour of the larger system causing denial of service and possibly other threats. Where the network is a distributed SCADA system, this could cause misbehaviour of the process that is under control.

Threat Escalation: If the firmware expects configuration that is present in Network A devices, but not Network B devices, then the device may experience degraded security, leading to threats of All Types.

11.3. Security Requirements

The security requirements here are a set of policies that mitigate the threats described in the previous section.

11.3.1. Security Requirement MFSR1: Monotonic Sequence Numbers

Only an actor with firmware installation authority is permitted to decide when device firmware can be installed. To enforce this rule, Manifests MUST contain monotonically increasing sequence numbers. Manifests MAY use UTC epoch timestamps to coordinate monotonically increasing sequence numbers across many actors in many locations. Devices MUST reject manifests with sequence numbers smaller than any onboard sequence number.

N.B. This is not a firmware version. It is a manifest sequence number. A firmware version may be rolled back by creating a new manifest for the old firmware version with a later sequence number.

Mitigates: Threat MFT1

11.3.2. Security Requirement MFSR2: Vendor, Device-type Identifiers

Devices MUST only apply firmware that is intended for them. Devices MUST know with fine granularity that a given update applies to their vendor, model, hardware revision, software revision. Human-readable identifiers are often error-prone in this regard, so unique identifiers SHOULD be used.

Mitigates: Threat MFT2

11.3.3. Security Requirement MFSR3: Best-Before Timestamps

Firmware MAY expire after a given time. Devices MAY provide a secure clock (local or remote). If a secure clock is provided and the Firmware manifest has a best-before timestamp, the device MUST reject the manifest if current time is larger than the best-before time.

Mitigates: Threat MFT3

11.3.4. Security Requirement MFSR4: Signed Payload Descriptor

All descriptive information about the payload MUST be signed. This MUST include:

- The location to store the payload
- The payload digest, in each state of installation (encrypted, plaintext, installed, etc.)
- The payload size
- The payload format
- Where to obtain the payload
- All instructions or parameters for applying the payload
- Any rules that identify whether or not the payload can be used on this device

Mitigates: Threats MFT5, MFT6, MFT7, MFT9

11.3.5. Security Requirement MFSR5: Cryptographic Authenticity

The authenticity of an update must be demonstrable. Typically, this means that updates must be digitally signed. Other cryptographic mechanisms are acceptable, such as MACs, or AEAD algorithms. Because the manifest contains information about how to install the update, the manifest's authenticity must also be demonstrable. To reduce the overhead required for validation, the manifest contains the digest of the firmware image, rather than a second digital signature. The authenticity of the manifest can be verified with a digital signature, the authenticity of the firmware image is tied to the manifest by the use of a fingerprint of the firmware image.

Mitigates: Threat MFT8

11.3.6. Security Requirement MFSR6: Rights Require Authenticity

If a device grants different rights to different actors, exercising those rights **MUST** be accompanied by proof of those rights, in the form of proof of authenticity. Authenticity mechanisms such as those required in MFSR5 are acceptable but need to follow the end-to-end security model.

For example, if a device has a policy that requires that firmware have both an Authorship right and a Qualification right and if that device grants Authorship and Qualification rights to different parties, such as an OEM and an Operator, respectively, then the firmware cannot be installed without proof of rights from both the OEM and the Operator.

Mitigates: MFT10

11.4. User Stories

User stories provide expected use cases. These are used to feed into usability requirements.

11.4.1. Use Case MFUC1: Installation Instructions

As an OEM for IoT devices, I want to provide my devices with additional installation instructions so that I can keep process details out of my payload data.

Some installation instructions might be:

- Specify a package handler

- Use a table of hashes to ensure that each block of the payload is validate before writing.
- Run post-processing script after the update is installed
- Do not report progress
- Pre-cache the update, but do not install
- Install the pre-cached update matching this manifest
- Install this update immediately, overriding any long-running tasks.

11.4.2. Use Case MFUC2: Reuse Local Infrastructure

As an Operator of IoT devices, I would like to tell my devices to look at my own infrastructure for payloads so that I can manage the traffic generated by firmware updates on my network and my peers' networks.

11.4.3. Use Case MFUC3: Modular Update

As an OEM of IoT devices, I want to divide my firmware into frequently updated and infrequently updated components, so that I can reduce the size of updates and make different parties responsible for different components.

11.4.4. Use Case MFUC4: Multiple Authorisations

As an Operator, I want to ensure the quality of a firmware update before installing it, so that I can ensure a high standard of reliability on my network. The OEM may restrict my ability to create firmware, so I cannot be the only authority on the device.

11.4.5. Use Case MFUC5: Multiple Payload Formats

As a OEM or Operator of devices, I want to be able to send multiple payload formats to suit the needs of my update, so that I can optimise the bandwidth used by my devices.

11.5. Usability Requirements

The following usability requirements satisfy the user stories listed above.

11.5.1. Usability Requirement MFUR1:

It must be possible to write additional installation instructions into the manifest.

Satisfies Use-Case MFUC1

11.5.2. Usability Requirement MFUR2:

It must be possible to redirect payload fetches. This applies where two manifests are used in conjunction. For example, an OEM manifest specifies a payload and signs it, and provides a URI for that payload. An Operator creates a second manifest, with a dependency on the first. They use this second manifest to override the URIs provided by the OEM, directing them into their own infrastructure instead.

Satisfies Use-Case MFUC2

11.5.3. Usability Requirement MFUR3:

It MUST be possible to link multiple manifests together so that a multi-component update can be described. This allows multiple parties with different permissions to collaborate in creating a single update for the IoT device, across multiple components.

Satisfies Use-Case MFUC2, MFUC3

11.5.4. Usability Requirement MFUR4:

It MUST be possible to sign a manifest multiple times so that signatures from multiple parties with different permissions can be required in order to authorise installation of a manifest.

Satisfies Use-Case MFUC4

11.5.5. Usability Requirement MFUR5:

The manifest format MUST accommodate any payload format that an operator or OEM wishes to use. Some examples of payload format would be:

- Binary
- Elf
- Differential

- Compressed
- Packed configuration

Satisfies Use-Case MFUC5

11.6. Manifest Fields

Each manifest field is anchored in a security requirement or a usability requirement. The manifest fields are described below and justified by their requirements.

11.6.1. Manifest Field: Timestamp

A monotonically increasing sequence number

Implements: Security Requirement MFSR1.

11.6.2. Manifest Field: Vendor ID Condition

Vendor IDs MUST be unique. This is to prevent similarly, or identically named entities from different geographic regions from colliding in their customer's infrastructure. Recommended practice is to use type 5 UUIDs with the vendor's domain name and the UUID DNS prefix. Other options include type 1 and type 4 UUIDs.

Implements: Security Requirement MFSR2, MFSR4.

11.6.3. Manifest Field: Class ID Condition

Class Identifiers MUST be unique within a Vendor ID. This is to prevent similarly, or identically named devices colliding in their customer's infrastructure. Recommended practice is to use type 5 UUIDs with the model, hardware revision, etc. and use the Vendor ID as the UUID prefix. Other options include type 1 and type 4 UUIDs. A device "Class" is defined as any device that can run the same firmware without modification. Classes MAY be implemented in a more granular way. Classes MUST NOT be implemented in a less granular way. Class ID can encompass model name, hardware revision, software revision. Devices MAY have multiple Class IDs.

Implements: Security Requirement MFSR2, MFSR4.

11.6.4. Manifest Field: Precursor Image Digest Condition

When a precursor image is required by the payload format, a precursor image digest condition MUST be present in the conditions list.

Implements: Security Requirement MFSR4

11.6.5. Manifest Field: Best-Before timestamp

This field tells a device the last application time. This is only usable in conjunction with a secure clock.

Implements Security Requirement MFSR3

11.6.6. Manifest Field: Payload Format

The format of the payload must be indicated to devices in an unambiguous way. This field provides a mechanism to describe the payload format, within the signed metadata.

Implements Security Requirement MFSR4, Usability Requirement MFUR5

11.6.7. Manifest Field: Storage Location

This field tells the device which component is being updated. The device can use this to establish which permissions are necessary and the physical location to use.

Implements Security Requirement MFSR4

11.6.8. Manifest Field: URIs

This field is a list of weighted URIs that the device uses to select where to obtain a payload.

Implements Security Requirement MFSR4

11.6.9. Manifest Field: Digests

This field is a map of digests, each for a separate stage of installation. This allows the target device to ensure authenticity of the payload at every step of installation.

Implements Security Requirement MFSR4

11.6.10. Manifest Field: Size

The size of the payload in bytes.

Implements Security Requirement MFSR4

11.6.11. Manifest Field: Signature

This is not strictly a manifest field. Instead, the manifest is wrapped by a standardised authentication container, such as a COSE or CMS signature object. The authentication container MUST support multiple actors and multiple authentications. If Message Authentication Codes (MAC) are used for authenticity, then devices MUST have unique pre-shared keys. This means that each device MUST receive an unique MAC.

Implements Security Requirement MFSR5, MFSR6, MFUR4

11.6.12. Manifest Field: Directives

A list of instructions that the device should execute, in order, when installing the payload.

Implements Usability Requirement MFUR1

11.6.13. Manifest Field: Aliases

A list of URI/Digest pairs. A device should build an alias table while parsing a manifest tree and treat any aliases as top-ranked URIs for the corresponding digest.

Implements Usability Requirement MFUR2

11.6.14. Manifest Field: Dependencies

A list of URI/Digest pairs that refer to other manifests by digest. The manifests that are linked in this way must be acquired and installed simultaneously in order to form a complete update.

Implements Usability Requirement MFUR3

12. References

12.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<https://www.rfc-editor.org/info/rfc2119>>.

12.2. Informative References

[RFC8240] Tschofenig, H. and S. Farrell, "Report from the Internet of Things Software Update (IoTSU) Workshop 2016", [RFC 8240](https://www.rfc-editor.org/info/rfc8240), DOI 10.17487/RFC8240, September 2017, <<https://www.rfc-editor.org/info/rfc8240>>.

[STRIDE] Microsoft, "The STRIDE Threat Model", January 2018.

12.3. URIs

[1] <mailto:suit@ietf.org>

Authors' Addresses

Brendan Moran
Arm Limited

EMail: Brendan.Moran@arm.com

Milosch Meriac
Arm Limited

EMail: Milosch.Meriac@arm.com

Hannes Tschofenig
Arm Limited

EMail: hannes.tschofenig@gmx.net