SUIT
Internet-Draft
Intended status: Informational
Expires: September 6, 2018

B. Moran
M. Meriac
H. Tschofenig
Arm Limited
March 05, 2018

A Firmware Update Architecture for Internet of Things Devices
draft-moran-suit-architecture-03

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.

Commented [DT1]: This sentence is hard to read. Change passive tense to active tense

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.

Commented [DT2]: Maybe "agnostic to the mechanism by which the firmware images and associated metadata are transported"?

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.

Commented [DT3]: This is a good question for the WG. Should this be included now? Or deferred to a future version?

Status of This Memo

Copyright Notice

Table of Contents

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~~ a firmware image. The manifest is protected against modification and provides information about the author.

Commented [DT4]: "the" incorrectly implies a device has only one monolithic firmware image

- 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. Firmware is the more universal term. Both terms are used in this document and are interchangeable.

Commented [DT5]: Which? "firmware" and "firmware image"? clarify

The following entities are used:

Commented [DT6]: Missing one: the entity that authorizes deployment of an image onto a specific device.

- 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 fileservers 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

Commented [DT7]: This list is redundant with the subsequent section headers. Not sure this list is needed here.

- Friendly to broadcast delivery

- Uses state-of-the-art security mechanisms

- Rollback attacks must be prevented.

Commented [DT8]: Be consistent

- High reliability

- Operates with a small bootloader

- Small Parsers

Commented [DT9]: Be consistent

- 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 5, is used to ensure that the device can verify firmware images and manifests produced by authorized authors.

The use of post-quantum secure signature mechanisms, such as hash-based signatures, should be explored. A mandatory-to-implement set of algorithms has to be defined offering a key length of 112-bit symmetric key or security or more, as outlined in Section 20 of RFC 7925. This corresponds to a 233 bit ECC key or a 2048 bit RSA key.

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

A device presented with an old, but valid manifest and firmware must not be tricked into installing such firmware since a vulnerability in the old firmware image may allow an attacker to gain control of the device.

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. One way to achieve this functionality is to provide a minimum of two storage locations for firmware and one bootable

Commented [DT10]: Elaborate… why do we care about broadcast (as opposed to multicast) at all?

Commented [DT11]: Why? This statement needs elaboration.

Commented [DT12]: This is unclear since the end of the message mentions "the update", but what is the relationship between "the message" and "the update"? Is an update the image? Is an update just a notification that something is available? This doesn't seem to be worded in a mechanism-agnostic way.

Commented [DT13]: By whom? The owner of a device? The WG before this draft is done?

Commented [DT14]: Meaning one of the two storage locations is bootable? Or in addition to the storage locations? Clarify wording.

location for firmware. An alternative approach is to use a 2nd stage bootloader with built~~d~~-in full featured firmware update functionality such that it is possible to return to the update process after power down.

Note: This is an implementation requirement rather than a requirement on the manifest format.

3.6. Operates with a small bootloader

The bootloader must be minimal, containing only flash support, cryptographic primitives and optionally a recovery mechanism. The recovery mechanism is used in case the update process failed and may include support for firmware updates over serial, USB or even a limited version of wireless connectivity standard like a limited Bluetooth Smart. Such a recovery mechanism must provide security at least at the same level as the full featured firmware update functionalities.

Commented [DT15]: I get why the mechanism needs to support small bootloaders. But if this is saying that all bootloaders (including on non-constrained devices) must necessarily be small, then it needs more elaboration on why.

The bootloader needs to verify the received manifest and to install the bootable firmware image. 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.

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.

Note: This is an implementation requirement.

3.7. Small Parsers

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

Commented [DT16]: This is an implementation requirement. I would say they should (not must) be minimal, and that the architecture must support small parsers.

3.8. Minimal impact on existing firmware formats

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

3.9. 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.

4. Claims

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

Commented [DT17]: Can't parse grammar here

Some example claims are:

- 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

Commented [DT18]: What's a "secondary" policy?

The baseline claims 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.

5. Architecture

We start the architectural description with the security model. It is based on end-to-end security. Figure 1 illustrates the security model where a firmware image and the corresponding manifest are created by an author and verified by the device. The firmware image is integrity protected and may be encrypted. The manifest is integrity protected and authenticated. When the author is ready to distribute the firmware image it is conveyed using some communication channel to the device, which will typically involve the use of untrusted storage. Examples of untrusted storage are FTP servers, Web servers or USB sticks.

Commented [DT19]: … and may be encrypted?

```
                           +-----------+
+--------+  Firmware Image  |           |   Firmware Image  +--------+
|        |  + Manifest      | Untrusted |   + Manifest      |        |
| Device |<-----------------| Storage   |<------------------| Author |
|        |                  |           |                   |        |
+--------+                  +-----------+                   +--------+
     ^                                                          *
     *                                                          *
     ************************************************************
                        End-to-End Security
```

Figure 1: End-to-End Security.

Commented [DT20]: This figure implies that the manifest exactly matches what the author submitted. If it requires countersigning, this will not be the case.

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.

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

Commented [DT21]: An IoT device cannot use a server to do this for it? Seems limiting compared to what the ACE WG is doing.

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 or a pre-shared key of a device.

Commented [DT22]: Disagree. If the owner of the device does the encryption as part of the counter-signing step, the author need not know anything of the sort.

There are different types of delivery modes, which are illustrated~~s~~ based on examples below.

There is an option for embedding a firmware image into a manifest. This is a useful approach for deployments where devices are not connected to the Internet and cannot contact a dedicated server for download of the firmware. It is also applicable when the firmware update happens via a USB stick or via Bluetooth Smart. Figure 2 shows this delivery mode graphically.

Commented [DT23]: Why is this more useful than (say) embedding the manifest into the firmware image? Perhaps you just mean a more generic “bundled with” as opposed to saying which is embedded in the other?

```
              /------------\                 /------------\
             /Manifest with \               /Manifest with \
             |attached      |               |attached      |
             \firmware image/               \firmware image/
              \------------/  +-----------+  \------------/
 +--------+                   |           |                 +--------+
 |        |<..................| Untrusted |<................|        |
 | Device |                   | Storage   |                 | Author |
 |        |                   |           |                 |        |
 +--------+                   +-----------+                 +--------+
```

Figure 2: Manifest with attached firmware.

Figure 3 shows an option for remotely updating a device where the device fetches the firmware image from some file server. The manifest itself is delivery independently and provides information about the firmware image(s) to download.

Commented [DT24]: Can’t parse grammar

Commented [DT25]: This presumes a pull mechanism, reword to be more agnostic as to protocol.

```
                                /------------\
                               /              \
                               |   Manifest   |
                               \              /
    +--------+                  \------------/                +--------+
    |        |<..............................................>|        |
    | Device |                                             -- | Author |
    |        |<-                                         ---  |        |
    +--------+  --                                     ---    +--------+
                  --                                 ---
                    ---                            ---
                       --       +-----------+    --
                         --     |           |  --
          /------------\   --   | Untrusted |<-    /------------\
         /              \    -- | Storage   |     /              \
         |   Firmware   |       |           |     |   Firmware   |
         \              /       +-----------+     \              /
          \------------/                           \------------/
```

Figure 3: Independent retrieval of the firmware image.

This architecture does not mandate a specific delivery mode but a solution must support both types.

Commented [DT26]: If both the manifest and the image are on untrusted storage, but they are not bundled together, which type is that?

6. 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?

Commented [DT27]: Meaning what? Give an example?

Commented [DT28]: How is this question different from the third question?

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 intend~~t~~ed to be applied to,

Commented [DT29]: Need more specific details in an information model (probably a separate draft). And it’s different from the manifest format which is a specific serialization.

- 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.

7. 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.

Commented [DT30]: The entire firmware, not a differential update?

```
   +--------+    +-----------------+      +------+
   | Author |    |Untrusted Storage|      |Device|
   +--------+    +-----------------+      +------+
     |                   |                     |
     | Create Firmware   |                     |
     |---------------    |                     |
     |              |    |                     |
     |<--------------    |                     |
     |                   |                     |
     | Upload Firmware   |                     |
     |------------------>|                     |
     |                   |                     |
     | Create Manifest   |                     |
     |----------------   |                     |
     |               |   |                     |
     |<---------------   |                     |
     |                   |                     |
     | Sign Manifest     |                     |
     |--------------     |                     |
     |             |     |                     |
     |<-------------      |                     |
     |                   |                     |
     | Upload Manifest   |                     |
     |------------------>|                     |
     |                   |                     |
     |                   |   Query Manifest    |
     |                   |<--------------------|
     |                   |                     |
     |                   |   Send Manifest     |
     |                   |-------------------->|
     |                   |                     |
     |                   |                     | Validate Manifest
     |                   |                     |------------------
     |                   |                     |                 |
     |                   |                     |<-----------------
     |                   |                     |
     |                   |  Request Firmware   |
     |                   |<--------------------|
     |                   |                     |
     |                   | Send Firmware       |
     |                   |-------------------->|
     |                   |                     |
     |                   |                     | Verify Firmware
     |                   |                     |---------------
     |                   |                     |              |
     |                   |                     |<--------------
     |                   |                     |
     |                   |                     | Store Firmware
     |                   |                     |--------------
     |                   |                     |             |
     |                   |                     |<-------------
     |                   |                     |
     |                   |                     | Reboot
     |                   |                     |-------
     |                   |                     |      |
     |                   |                     |<------
     |                   |                     |
     |                   |                     | Bootloader validates
     |                   |                     | Firmware
     |                   |                     |----------------------
     |                   |                     |                     |
     |                   |                     |<---------------------
     |                   |                     |
     |                   |                     | Bootloader activates
     |                   |                     | Firmware
     |                   |                     |----------------------
     |                   |                     |                     |
     |                   |                     |<---------------------
```

```
    |                   |                     |
    |                   |                     | Bootloader transfers
    |                   |                     | control to new Firmware
    |                   |                     |----------------------
    |                   |                     |                     |
    |                   |                     |<---------------------
    |                   |                     |
```

Figure 4: Example Flow for a Firmware Upate.

Commented [DT31]: typo

8. IANA Considerations

This document does not require any actions by IANA.

9. 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~~in [RFC8240]. This document (and associated specifications) offer a standardized firmware manifest format providing end-to-end security from the author to the device.

Commented [DT32]: No it doesn't, only a separate document does.

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 digitial signature protecting the manifest.

- incentives for manufacturers to offer a firmware update mechanism as part of their IoT products.

Commented [DT33]: typo

10. 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

Commented [DT34]: This section should be deleted if this document becomes a WG doc.

11. Acknowledgements

We would like to thank 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

- Marti Bolivar

- Andrzej Puzdrowski

- Markus Gueller

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 and their reviews.

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, 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

Appendix A. Threat Model, User Stories, Security Requirements, and Usability Requirements

Commented [DT35]: Why is this in an appendix rather than in the body? At least add a note about why here in the text. Personally, I'd prefer it be in the body.

A.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 one of the following types:

- Spoofing Identity

- Tampering with data

- Repudiation

- Information disclosure

- Denial of service

- Elevation of privilege

This threat model only covers elements related to the transport of firmware updates. It explicitly does not cover threats outside of the transport of firmware updates. For example, threats to an IoT device due to physical access are out of scope.

A.2. Threat Descriptions

A.2.1. Threat MFT1: Old Firmware

Classification: Escalation of Privilege

Commented [DT36]: Elevation. Many more occurrences below.

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.

Commented [DT37]: This wording seems to incorrectly imply they have to be bundled in this threat

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

Mitigated by: MFSR1

A.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.

Commented [DT38]: Does this threat really only apply to a push-based mechanism? I think not. Need to fix wording.

Mitigated by: MFSR2

A.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.

Commented [DT39]: Same comment here on “sends” and “with”

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

Mitigated by: MFSR3

A.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.

Mitigated by: MFSR4

A.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.

Mitigated by: MFSR4

A.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.

Mitigated by: MFSR4

A.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.

Mitigated by: MFSR4

A.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.

Mitigated by: MFSR5

A.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.

Mitigated by: MFSR4

A.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

Commented [DT40]: use consistent capitalization in section headings

Commented [DT41]: undefined term

Commented [DT42]: I think it would be more correct to say
“The owner or operator of a device needs to approve firmware for their devices”
since the network owner and the device owner might be different.

Same issue applies to the rest of this section 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.

Mitigated by: MFSR6

A.2.11. Threat MFT11: Reverse Engineering Of Firmware Image for Vulnerability Analysis

Classification: All Types

An attacker wants to mount an attack on an IoT device. To prepare the attack he or she retrieves the provided firmware image and performs reverse engineering of the firmware image to analyze it for specific vulnerabilities.

Mitigated by: MFSR7

A.3. Security Requirements

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

A.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 increasting sequence numbers. Manifests MAY use UTC epoch timestamps to coordinate monotonically increasting sequence numbers across many actors in many locations. Devices MUST reject manifests with sequence numbers smaller than any onboard sequence number.

Commented [DT43]: typo

Commented [DT44]: typo

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 Implemented by: Manifest Field: Timestamp

A.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 Implemented by: Manifest Fields: Vendor ID Condition, Class ID Condition

A.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 the current time is larger than the best-before time.

Mitigates: Threat MFT3 Implemented by: Manifest Field: Best-Before timestamp condition

A.3.4. Security Requirement MFSR4: Signed Payload Descriptor

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

- The type of payload (which may be independent of format)

Commented [DT45]: meaning what?

- 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 MFT4, MFT5, MFT6, MFT7, MFT9 Implemented by: Manifest Fields: Vendor ID Condition, Class ID Condition, Precursor Image Digest Condition, Payload Format, Storage Location, URIs, Digests, Size

A.3.5. Security Requirement MFSR5: Cryptographic Authenticity

The authenticity of an update must be demonstrable. Typically, this means that updates must be digitally signed. 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 digitial 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.

Commented [DT46]: typo

Mitigates: Threat MFT8 Implemented by: Signature

A.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.

Commented [DT47]: which is specified where?

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 Implemented by: Signature

A.3.7. Security Requirement MFSR7: Firmware encryption

Firmware images must be encrypted to prevent third parties, including attackers, from reading the content of the firmware image and to reverse engineer the code.

Mitigates: MFT11 Implemented by: Manifest Field: Content Key Distribution Method

A.4. User Stories

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

A.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.

Satisfied by: MFUR1

A.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.

Satisfied by: MFUR2, MFUR3

A.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.

Satisfied by: MFUR3

A.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.

Satisfied by: MFUR4

A.4.5. Use Case MFUC5: Multiple Payload Formats

As an 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.

Commented [DT48]: Not sure what this means. Elaborate.

Satisfied by: MFUR5

A.4.6. Use Case MFUC6: IP Protection

As an OEM or developer for IoT devices, I want to protect the IP contained in the firmware image, such as the utilized algorithms. The need for protecting IP may have also been imposed on my due to the use of some third party code libraries.

Commented [DT49]: Can’t parse grammar

Satisfied by: MFSR7

A.5. Usability Requirements

The following usability requirements satisfy the user stories listed above.

A.5.1. Usability Requirement MFUR1

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

Satisfies: Use-Case MFUC1 Implemented by: Manifest Field: Directives

A.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 Implemented by: Manifest Field: Aliases

A.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 Implemented by: Manifest Field: Dependencies

A.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 Implemented by: COSE Signature (or similar)

A.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 Implemented by: Manifest Field: Payload Format

A.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.

A.6.1. Manifest Field: Timestamp

A monotonically increasing sequence number. For convenience, a timestamp implements the requirement of a monotonically increasing sequence number. This allows global synchronisation of sequence numbers without any additional management.

Commented [DT50]: This has the problem that it requires synchronized clocks, which seems too burdensome to require for all constrained devices. If sequence number were mandatory, and timestamp were optional, it would be better.

Implements: Security Requirement MFSR1.

A.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.

Commented [DT51]: This seems like it needs WG discussion.
Other existing formats use other types of vendor ID (for example, UEFI uses 16-bit vendor IDs assigned by USB-IF).

Implements: Security Requirement MFSR2, MFSR4.

A.6.3. Manifest Field: Class ID Condition

Commented [DT52]: Undefined term

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.

A.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

A.6.5. Manifest Field: Best-Before timestamp condition

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

Commented [DT53]: Meaning the field is not mandatory? Seems like the information model needs to say which fields are mandatory, which are optional. Similarly, whether they can occur multiple times or not.

Implements: Security Requirement MFSR3

A.6.6. Manifest Field: Payload Format

The format of the payload must be indicated to devices is 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

A.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

A.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

A.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

A.6.10. Manifest Field: Size

The size of the payload in bytes.

Implements: Security Requirement MFSR4

A.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.

Implements: Security Requirement MFSR5, MFSR6, MFUR4

A.6.12. Manifest Field: Directives

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

Implements: Usability Requirement MFUR1

A.6.13. Manifest Field: Aliases

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

Implements: Usability Requirement MFUR2

A.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

A.6.15. Manifest Field: Content Key Distribution Method

Encrypting firmware images requires symmetric content encryption keys. Since there are several methods to protect or distribute the symmetric content encryption keys, the manifest contains a field for the Content Key Distribution Method. One examples for such a Content Key Distribution Method is the usage of Key Tables, pointing to content encryption keys, which themselves are encrypted using the public keys of devices.

Commented [DT54]: Fix grammar

Implements: Security Requirement MFSR7.

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