SUIT B. Moran
Internet-Draft H. Tschofenig
Intended status: Standards Track Arm Limited
Expires: July 22, 2019 H. Birkholz
Fraunhofer SIT
January 18, 2019

Firmware Updates for Internet of Things Devices - An Information Model for Manifests
draft-ietf-suit-information-model-02

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.

One component of such a firmware update is the meta-data, or manifest, that describes the firmware image(s) and offers appropriate protection. This document describes all the information that must be present in the manifest.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

The information model describes all the information elements required to secure firmware updates of IoT devices from the threats described in Section 4.1 and enable the user stories captured in Section 4.4. These threats and user stories are not intended to be an exhaustive list of the threats against IoT devices, nor of the possible use cases of firmware update; instead they are intended to describe the threats against firmware update in isolation and provide sufficient motivation to provide information elements that cover a wide range of use cases. The information model does not define the encoding, ordering, or structure of information elements, only their semantics.

Because the information model covers a wide range of user stories and a wide range of threats, not all information elements apply to all scenarios. As a result, many information elements could be considered optional to implement and optional to use, depending on which threats exist in a particular system and which use cases are required. Elements marked as mandatory provide baseline security and usability properties that are expected to be required for most applications. Those elements are mandatory to implement and mandatory to use. Elements marked as recommended provide important security or usability properties that are needed on most devices. Elements marked as optional enable security or usability properties that are useful in some applications.

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 terms defined in [I-D.ietf-suit-architecture]. The term 'Operator' refers to both, Device and Network Operator.

3. Manifest Information Elements

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

3.1. Manifest Element: version identifier of the manifest structure

An identifier that describes which iteration of the manifest format is contained in the structure.

This element is MANDATORY and must be present in order to allow devices to identify the version of the manifest data model that is in use.

3.2. Manifest Element: Monotonic Sequence Number

A monotonically increasing sequence number. For convenience, the monotonic sequence number MAY be a UTC timestamp. This allows global synchronisation of sequence numbers without any additional management.

This element is MANDATORY and is necessary to prevent malicious actors from reverting a firmware update against the wishes of the relevant authority.

Implements: Security Requirement MFSR1.

3.3. Manifest Element: 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.

This ID is RECOMMENDED and helps to distinguish between identically named products from different vendors.

Implements: Security Requirement MFSR2, MFSR4f.

3.3.1. Example: Domain Name-based UUIDs

Vendor A creates a UUID based on their domain name:

```
vendorId = UUID5(DNS, "vendor-a.com")
```

Because the DNS infrastructure prevents multiple registrations of the same domain name, this UUID is guaranteed to be unique. Because the domain name is known, this UUID is reproducible. Type 1 and type 4 UUIDs produce similar guarantees of uniqueness, but not reproducibility.

3.4. Manifest Element: Class ID Condition

A device "Class" is defined as any device that can accept the same firmware update without modification. 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. 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.

Note Well: Class ID is not a human-readable element. It is intended for match/mismatch use only.

This ID is RECOMMENDED and allows devices to determine applicability of a firmware in an unambiguous way.

Implements: Security Requirement MFSR2, MFSR4f.

3.4.1. Example 1: Different Classes

Vendor A creates product Z and product Y. The firmware images of products Z and Y are not interchangeable. Vendor A creates UUIDs as follows:

- vendorId = UUID5(DNS, "vendor-a.com")

- ZclassId = UUID5(vendorId, "Product Z")

- YclassId = UUID5(vendorId, "Product Y")

This ensures that Vendor A’s Product Z cannot install firmware for Product Y and Product Y cannot install firmware for Product Z.

3.4.2. Example 2: Upgrading Class ID

Vendor A creates product X. Later, Vendor A adds a new feature to product X, creating product X v2. Product X requires a firmware update to work with firmware intended for product X v2.

Vendor A creates UUIDs as follows:

- vendorId = UUID5(DNS, "vendor-a.com")

- XclassId = UUID5(vendorId, "Product X")

- Xv2classId = UUID5(vendorId, "Product X v2")

When product X receives the firmware update necessary to be compatible with product X v2, part of the firmware update changes the class ID to Xv2classId.

3.4.3. Example 3: Shared Functionality

Vendor A produces two products, product X and product Y. These components share a common core (such as an operating system), but have different applications. The common core and the applications can be updated independently. To enable X and Y to receive the same common core update, they require the same class ID. To ensure that only product X receives application X and only product Y receives application Y, product X and product Y must have different class IDs. The vendor creates Class IDs as follows:

- vendorId = UUID5(DNS, "vendor-a.com")

- XclassId = UUID5(vendorId, "Product X")

- YclassId = UUID5(vendorId, "Product Y")

- CommonClassId = UUID5(vendorId, "common core")

Product X matches against both XclassId and CommonClassId. Product Y matches against both YclassId and CommonClassId.

3.5. Manifest Element: 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. The precursor image may be installed or stored as a candidate.

This element is MANDATORY for differential updates. Otherwise, it is not needed.

Implements: Security Requirement MFSR4e

3.6. Manifest Element: Required Image Version List

When a payload applies to multiple versions of a firmware, the required image version list specifies which versions must be present for the update to be applied. This allows the update author to target specific versions of firmware for an update, while excluding those to which it should not be applied.

Where an update can only be applied over specific predecessor versions, that version MUST be specified by the Required Image Version List.

This element is OPTIONAL.

Implements: MFUR7

3.7. Manifest Element: Best-Before timestamp condition

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

This element is OPTIONAL and MAY enable use cases where a secure clock is provided and firmware is intended to expire regularly.

Implements: Security Requirement MFSR3

3.8. Manifest Element: Payload Format

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

This element is MANDATORY and MUST be present to enable devices to decode payloads correctly.

Implements: Security Requirement MFSR4a, Usability Requirement MFUR5

3.9. Manifest Element: Processing Steps

A list of all payload processors necessary to process a nested format and any parameters needed by those payload processors. Each Processing Step SHOULD indicate the expected digest of the payload after the processing is complete. Processing steps are distinct from Directives in that Directives apply to the manifest as a whole, whereas Processing Steps apply to an individual payload and provide instructions on how to unpack it.

Implements: Usability Requirement MFUR6

3.10. Manifest Element: Storage Location

This element 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.

This element is MANDATORY and MUST be present to enable devices to store payloads to the correct location.

Implements: Security Requirement MFSR4b

3.10.1. Example 1: Two Storage Locations

A device supports two components: an OS and an application. These components can be updated independently, expressing dependencies to ensure compatibility between the components. The firmware authority chooses two storage identifiers:

- OS

- APP

3.10.2. Example 2: File System

A device supports a full filesystem. The firmware authority chooses to make the storage identifier the path at which to install the payload. The payload may be a tarball, in which case, it unpacks the tarball into the specified path.

3.10.3. Example 3: Flash Memory

A device supports flash memory. The firmware authority chooses to make the storage identifier the offset where the image should be written.

3.11. Manifest Element: Component Identifier

In a heterogeneous storage architecture, a storage identifier is insufficient to identify where and how to store a payload. To resolve this, a component identifier indicates which part of the storage architecture is targeted by the payload. In a homogeneous storage architecture, this element is unnecessary.

This element is OPTIONAL and only necessary in heterogeneous storage architecture devices.

Implements: MFUR3

3.12. Manifest Element: URIs

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

This element is OPTIONAL and only needed when the target device does not intrinsically know where to find the payload.

Note: Devices will typically require URIs.

Implements: Security Requirement MFSR4c

3.13. Manifest Element: Payload Digest

This element contains the digest of the payload. This allows the target device to ensure authenticity of the payload. It MUST be possible to specify more than one payload digest, indexed by Manifest Element: XIP Address.

This element is MANDATORY and fundamentally necessary to ensure the authenticity and integrity of the payload.

Implements: Security Requirement MFSR4d, Usability Requirement MFUR8

3.14. Manifest Element: Size

The size of the payload in bytes.

This element is MANDATORY and informs the target device how big of a payload to expect. Without it, devices are exposed to some classes of denial of service attack.

Implements: Security Requirement MFSR4d

3.15. Manifest Element: Signature

This is not strictly a manifest element. 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.

This element is MANDATORY and represents the foundation of all security properties of the manifest.

Implements: Security Requirement MFSR5, MFSR6, MFUR4

3.16. Manifest Element: Directives

A list of instructions that the device should execute, in order, when processing the manifest. This information is distinct from the information necessary to process a payload (Processing Steps) and applies to the whole manifest including all payloads that it references. Directives include information such as update timing (For example, install only on Sunday, at 0200), procedural considerations (for example, shut down the equipment under control before executing the update), pre and post-installation steps (for example, run a script).

This element is OPTIONAL and enables some use cases.

Implements: Usability Requirement MFUR1

3.17. Manifest Element: Aliases

A list of Digest/URI 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.

This element is OPTIONAL and enables some use cases.

Implements: Usability Requirement MFUR2

3.18. Manifest Element: Dependencies

A list of Digest/URI 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.

This element is MANDATORY to use in deployments that include both multiple authorities and multiple payloads.

Implements: Usability Requirement MFUR3

3.19. Manifest Element: 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 element 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. This MAY be included in a decryption step contained in Processing Steps.

This element is MANDATORY to use for encrypted payloads,

Implements: Security Requirement MFSR7.

3.20. Manifest Element: XIP Address

In order to support XIP systems with multiple possible base addresses, it is necessary to specify which address the payload is linked for.

For example a microcontroller may have a simple bootloader that chooses one of two images to boot. That microcontroller then needs to choose one of two firmware images to install, based on which of its two images is older.

Implements: MFUR8

3.21. Manifest Element: Load-time metadata

Manifest Element: Boot-time metadata ## Manifest Element: Payload

4. Motivation for Manifest Fields

The following sub-sections describe the threat model, user stories, security requirements, and usability requirements.

4.1. Threat Model

The following sub-sections aim 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

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.

4.2. Threat Descriptions

4.2.1. Threat MFT1: Old Firmware

Classification: Elevation 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.

Mitigated by: MFSR1

4.2.2. 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.

Mitigated by: MFSR2

Example:

Suppose that two vendors, Vendor A and Vendor B, adopt the same trade name in different geographic regions, and they both make products with the same names, or product name matching is not used. This causes firmware from Vendor A to match devices from Vendor B.

If the vendors are the firmware authorities, then devices from Vendor A will reject images signed by Vendor B since they use different credentials. However, if both devices trust the same firmware authority, then, devices from Vendor A could install firmware intended for devices from Vendor B.

4.2.3. Threat MFT3: Offline device + Old Firmware

Classification: Elevation 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.

Mitigated by: MFSR3

4.2.4. 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 elevation of privilege and potentially expand this to all types of threat.

Mitigated by: MFSR4a

4.2.5. 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 elevation of privilege and potentially expand this to all types of threat.

Mitigated by: MFSR4b

4.2.6. 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: MFSR4c

4.2.7. Threat MFT7: Payload Verification on Boot

Classification: Elevation of Privilege

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.

Threat Escalation: If the attacker is able to exploit a known vulnerability, or if the attacker can supply their own firmware, then this threat can be escalated to ALL TYPES.

Mitigated by: MFSR4d

4.2.8. Threat MFT8: Unauthenticated Updates

Classification: Elevation of Privilege

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

Threat Escalation: If the attacker is able to exploit a known vulnerability, or if the attacker can supply their own firmware, then this threat can be escalated to ALL TYPES.

Mitigated by: MFSR5

4.2.9. 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 elevation of privilege and potentially expand this to all types of threat.

Mitigated by: MFSR4e

4.2.10. Threat MFT10: Unqualified Firmware

Classification: Denial of Service, Elevation 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.

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

Mitigated by: MFSR6, MFSR8

4.2.10.1. Example 1: Multiple Network Operators with a Single Device Operator

In this example let us assume that Device Operators expect the rights to create firmware but that Network Operators expect the rights to qualify firmware as fit-for-purpose on their networks. Additionally assume that an Device Operators manage devices that can be deployed on any network, including Network A and B in our example.

An attacker may obtain a manifest for a device on Network A. Then, this attacker sends that manifest to a device on Network B. Because Network A and Network B are under control of different Operators, and the firmware for a device on Network A has not been qualified to be deployed on Network B, the target device on Network B is now in violation of the Operator B’s policy and may get disabled by this unqualified, but signed firmware.

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

4.2.10.2. Example 2: Single Network Operator with Multiple Device Operators

Multiple devices that interoperate are used on the same network and communicate with each other. Some devices are manufactured and managed by Device Operator A and other devices by Device Operator B. A new firmware is released by Device Operator A that breaks compatibility with devices from Device Operator B. An attacker sends the new firmware to the devices managed by Device Operator A 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.

4.2.11. 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

4.2.12. Threat MFT12: Overriding Critical Manifest Elements

Classification: Elevation of Privilege

An authorised actor, but not the firmware authority, uses an override mechanism (MFUS2) to change an information element in a manifest signed by the firmware authority. For example, if the authorised actor overrides the digest and URI of the payload, the actor can replace the entire payload with a payload of their choice.

Threat Escalation: By overriding elements such as payload installation instructions or firmware digest, this threat can be escalated to all types.

Mitigated by: MFSR8

4.2.13. Threat MFT13: Manifest Element Exposure

Classification: Information Disclosure

A third party may be able to extract sensitive information from the manifest.

Mitigated by: MFSR9

4.3. Security Requirements

The security requirements here are a set of policies that mitigate the threats described in Section 4.1.

4.3.1. 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. If UTC epoch timestamps are used, they MUST NOT be treated as times, they MUST be treated only as sequence numbers. Devices MUST reject manifests with sequence numbers smaller than any onboard sequence number.

Note: 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 Element: Monotonic Sequence Number

4.3.2. 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 Elements: Vendor ID Condition, Class ID Condition

4.3.3. 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

Implemented by: Manifest Element: Best-Before timestamp condition

4.3.4. Security Requirement MFSR5: Cryptographic Authenticity

The authenticity of an update must be demonstrable. Typically, this means that updates must be digitally authenticated. 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 or Message Authentication Code, the authenticity of the firmware image is tied to the manifest by the use of a digest of the firmware image.

Mitigates: Threat MFT8

Implemented by: Signature, Payload Digest

4.3.5. Security Requirement MFSR4a: Authenticated Payload Type

The type of payload (which may be independent of format) MUST be authenticated. For example, the target must know whether the payload is XIP firmware, a loadable module, or serialized configuration data.

Mitigates: MFT4

Implemented by: Manifest Elements: Payload Format, Storage Location

4.3.6. Security Requirement MFSR4b: Authenticated Storage Location

The location on the target where the payload is to be stored MUST be authenticated.

Mitigates: MFT5

Implemented by: Manifest Elements: Storage Location

4.3.7. Security Requirement MFSR4c: Authenticated Remote Resource Location

The location where a target should find a payload MUST be authenticated.

Mitigates: MFT6

Implemented by: Manifest Elements: URIs

4.3.8. Security Requirement MFSR4d: Secure Boot

The target SHOULD verify firmware at time of boot. This requires authenticated payload size, and digest.

Mitigates: MFT7

Implemented by: Manifest Elements: Payload Digest, Size

4.3.9. Security Requirement MFSR4e: Authenticated precursor images

If an update uses a differential compression method, it MUST specify the digest of the precursor image and that digest MUST be authenticated.

Mitigates: MFT9

Implemented by: Manifest Elements: Precursor Image Digest Condition

4.3.10. Security Requirement MFSR4f: Authenticated Vendor and Class IDs

The identifiers that specify firmware compatibility MUST be authenticated to ensure that only compatible firmware is installed on a target device.

Mitigates: MFT2

Implemented By: Manifest Elements: Vendor ID Condition, Class ID Condition

4.3.11. Security Requirement MFSR4f: Authenticated Vendor and Class IDs

The identifiers that specify firmware compatibility MUST be authenticated to ensure that only compatible firmware is installed on a target device.

Mitigates: MFT2

Implemented By: Manifest Elements: Vendor ID Condition, Class ID Condition

4.3.12. 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 a Device Operator and a Network Operator, respectively, then the firmware cannot be installed without proof of rights from both the Device and the Network Operator.

Mitigates: MFT10

Implemented by: Signature

4.3.13. Security Requirement MFSR7: Firmware encryption

The manifest information model must enable encrypted payloads. Encryption helps to prevent third parties, including attackers, from reading the content of the firmware image. This can protect against confidential information disclosures and discovery of vulnerabilities through reverse engineering. Therefore the manifest must convey the information required to allow an intended recipient to decrypt an encrypted payload.

Mitigates: MFT11

Implemented by: Manifest Element: Content Key Distribution Method

4.3.14. Security Requirement MFSR8: Access Control Lists

If a device grants different rights to different actors, then an exercise of those rights must be validated against a list of rights for the actor. This typically takes the form of an Access Control List (ACL). ACLs are applied to two scenarios:

1. An ACL decides which elements of the manifest may be overridden and by which actors.

2. An ACL decides which component identifier/storage identifier pairs can be written by which actors.

Mitigates: MFT12, MFT10

Implemented by: Client-side code, not specified in manifest.

4.3.15. Security Requirement MFSR9: Encrypted Manifests

It must be possible to encrypt part or all of the manifest. This may be accomplished with either transport encryption or with at-rest encryption, for example COSE_Encrypt.

Mitigates: MFT13

Implemented by: TLS/COSE

4.4. User Stories

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

4.4.1. Use Case MFUS1: Installation Instructions

As an Device Operator, 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:

- Use a table of hashes to ensure that each block of the payload is validate before writing.

- 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

4.4.2. Use Case MFUS2: Override Non-Critical Manifest Elements

As a Network Operator, I would like to be able to override the non-critical information in the manifest so that I can control my devices more precisely. This assumes that the Device Operator delegated rights about the device to the Network Operator.

Some examples of potentially overridable information:

- URIs: this allows the Network Operator to direct devices to their own infrastructure in order to reduce network load.

- Conditions: this allows the Network Operator to pose additional constraints on the installation of the manifest.

- Directives: this allows the Network Operator to add more instructions such as time of installation.

- Processing Steps: If an intermediary performs an action on behalf of a device, it may need to override the processing steps. It is still possible for a device to verify the final content and the result of any processing step that specifies a digest. Some processing steps should be non-overridable.

Satisfied by: MFUR2, MFUR3

4.4.3. Use Case MFUS3: Modular Update

As an Operator, 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

4.4.4. Use Case MFUS4: Multiple Authorisations

As a Device Operator, I want to ensure the quality of a firmware update before installing it, so that I can ensure interoperability of all devices in my product family. I want to restrict the ability to make changes to my devices to require my express approval.

Satisfied by: MFUR4, MFSR8

4.4.5. Use Case MFUS5: Multiple Payload Formats

As an Operator, 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.

Satisfied by: MFUR5

4.4.6. Use Case MFUS6: Prevent Confidential Information Disclosures

As an firmware author, I want to prevent confidential information from being disclosed during firmware updates. It is assumed that channel security is adequate to protect the manifest itself against information disclosure.

Satisfied by: MFSR7

4.4.7. Use Case MFUS7: Prevent Devices from Unpacking Unknown Formats

As a Device Operator, I want devices to determine whether they can process a payload prior to downloading it.

In some cases, it may be desirable for a third party to perform some processing on behalf of a target. For this to occur, the third party MUST indicate what processing occurred and how to verify it against the Trust Provisioning Authority’s intent.

This amounts to overriding Processing Steps and URIs.

Satisfied by: MFUR6, MFUR2

4.4.8. Use Case MFUS8: Specify Version Numbers of Target Firmware

As a Device Operator, I want to be able to target devices for updates based on their current firmware version, so that I can control which versions are replaced with a single manifest.

Satisfied by: MFUR7

4.4.9. Use Case MFUS9: Enable Devices to Choose Between Images

As a developer, I want to be able to sign two or more versions of my firmware in a single manifest so that I can use a very simple bootloader that chooses between two or more images that are executed in-place.

Satisfied by: MFUR8

4.4.10. Use Case MFUS10: Secure Boot Using Manifests

As a signer for both secure boot and firmware deployment, I would like to use the same signed document for both tasks so that my data size is smaller, I can share common code, and I can reduce signature verifications.

Satisfied by: MFUR9

4.4.11. Use Case MFUS11: Decompress on Load

As a developer of firmware for a run-from-RAM device, I would like to use compressed images and to indicate to the bootloader that I am using a compressed image in the manifest so that it can be used with secure boot.

Satisfied by: MFUR10

4.4.12. Use Case MFUS12: Payload in Manifest

As an operator of a constrained network, I would like to be able to send a small payload in the same packet as the manifest so that I can reduce network traffic.

Satisfied by: MFUR11

4.4.13. Use Case MFUS13: Simple Parsing

As a developer for constrained devices, I want a low complexity library for processing updates so that I can fit more application code on my device.

Satisfied by: MFUR12

4.5. Usability Requirements

The following usability requirements satisfy the user stories listed above.

4.5.1. Usability Requirement MFUR1

It must be possible to provide all information necessary for the processing of a manifest into the manifest.

Satisfies: User story MFUS1

Implemented by: Manifest Element: Directives

4.5.2. Usability Requirement MFUR2

It must be possible to redirect payload fetches. This applies where two manifests are used in conjunction. For example, a Device Operator creates a manifest specifying a payload and signs it, and provides a URI for that payload. A Network Operator creates a second manifest, with a dependency on the first. They use this second manifest to override the URIs provided by the Device Operator, directing them into their own infrastructure instead. Some devices may provide this capability, while others may only look at canonical sources of firmware. For this to be possible, the device must fetch the payload, whereas a device that accpets payload pushes will ignore this feature.

Satisfies: User story MFUS2

Implemented by: Manifest Element: Aliases

4.5.3. Usability Requirement MFUR3

It must be possible express the requirement to install one or more payloads from one or more authorities so that a multi-payload 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.

This requirement effectively means that it must be possible to construct a tree of manifests on a multi-image target.

Because devices can be either HeSA or HoSA both the storage system and the storage location within that storage system must be possible to specify. In a HoSA device, the payload location may be as simple as an address, or a file path. In a HeSA device, the payload location may be scoped by a component identifier. It is expedient to consider that all HoSA devices are HeSA devices with a single component.

4.5.3.1. Example 1: Multiple Microcontrollers

An IoT device with multiple microcontrollers in the same physical device (HeSA) will likely require multiple payloads with different component identifiers.

4.5.3.2. Example 2: Code and Configuration

A firmware image can be divided into two payloads: code and configuration. These payloads may require authorizations from different actors in order to install (see MFSR6 and MFSR8). This structure means that multiple manifests may be required, with a dependency structure between them.

4.5.3.3. Example 3: Multiple Chunks

A firmware image can be divided into multiple functional blocks for separate testing and distribution. This means that code would need to be distributed in multiple payloads. For example, this might be desirable in order to ensure that common code between devices is identical in order to reduce distribution bandwidth.

Satisfies: User story MFUS2, MFUS3

Implemented by Manifest Element: Dependencies, StorageIdentifier, ComponentIdentifier

4.5.4. 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: User story MFUS4

Implemented by: COSE Signature (or similar)

4.5.5. Usability Requirement MFUR5

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

- Binary

- Elf

- Differential

- Compressed

- Packed configuration

- Intel HEX

- S-Record

Satisfies: User story MFUS5

Implemented by: Manifest Element: Payload Format

4.5.6. Usability Requirement MFUR6

The manifest format must accommodate nested formats, announcing to the target device all the nesting steps and any parameters used by those steps.

Satisfies: User story MFUS6

Implemented by: Manifest Element: Processing Steps

4.5.7. Usability Requirement MFUR7

The manifest format must provide a method to specify multiple version numbers of firmware to which the manifest applies, either with a list or with range matching.

Satisfies: User story MFUS8

Implemented by: Manifest Element: Required Image Version List

4.5.8. Usability Requirement MFUR8

The manifest format must provide a mechanism to list multiple equivalent payloads by Execute-In-Place Installation Address, including the payload digest and, optionally, payload URIs.

Satisfies: User story MFUS9

Implemented by: Manifest Element: XIP Address

4.5.9. Usability Requirement MFUR9: Bootable Manifest

It must be possible to describe a bootable system with a manifest on both Execute-In-Place microcontrollers and on complex operating systems. This requires the manifest to specify the digest of each statically linked storage location. In addition, the manifest must be able to express metadata used by the bootloader, such as a kernel command-line.

Satisfies: User story MFUS10

Implemented by: Manifest Element: Boot-time Metadata

4.5.10. Usability Requirement MFUR10: Load-Time Information

It must be possible to specify additional metadata for load time processing of a payload, such as load-address, and compression algorithm.

N.B. load comes before boot.

Satisfies: User Story MFUS11

Implemented by: Manifest Element: Load-time Metadata

4.5.11. Usability Requirement MFUR11: Payload in Manifest Superstructure

It must be possible to place a payload in the same structure as the manifest. This typically places the payload in the same packet as the manifest.

Satisfies: User Story MFUS12

Implemented by: Manifest Element: Payload

4.5.12. Usability Requirement MFUR12: Simple Parsing

The structure of the manifest must be simple to parse, without need for a general-purpose parser.

Satisfies: User Story MFUS13

Implemented by: N/A

5. Security Considerations

Security considerations for this document are covered in Section 4.

6. IANA Considerations

This document does not require any actions by IANA.

7. Acknowledgements

We would like to thank our working group chairs, Dave Thaler, Russ Housley and David Waltermire, for their review comments and their support.

We would like to thank the participants of the 2018 Berlin SUIT Hackathon and the June 2018 virtual design team meetings for their discussion input. In particular, we would like to thank Koen Zandberg, Emmanuel Baccelli, Carsten Bormann, David Brown, Markus Gueller, Frank Audun Kvamtro, Oyvind Ronningstad, Michael Richardson, Jan-Frederik Rieckers Francisco Acosta, Anton Gerasimov, Matthias Waehlisch, Max Groening, Daniel Petry, Gaetan Harter, Ralph Hamm, Steve Patrick, Fabio Utzig, Paul Lambert, Benjamin Kaduk, Said Gharout, and Milen Stoychev.

8. References

8.1. Normative References

[I-D.ietf-suit-architecture]
Moran, B., Meriac, M., Tschofenig, H., and D. Brown, "A Firmware Update Architecture for Internet of Things Devices", draft-ietf-suit-architecture-02 (work in progress), January 2019.

[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>.

8.2. Informative References

[RFC4122] Leach, P., Mealling, M., and R. Salz, "A Universally Unique IDentifier (UUID) URN Namespace", RFC 4122, DOI 10.17487/RFC4122, July 2005, <https://www.rfc-editor.org/info/rfc4122>.

[STRIDE] Microsoft, "The STRIDE Threat Model", May 2018, <https://msdn.microsoft.com/en-us/library/ee823878(v=cs.20).aspx>.

8.3. URIs

[1] mailto:suit@ietf.org

[2] https://www1.ietf.org/mailman/listinfo/suit

[3] https://www.ietf.org/mail-archive/web/suit/current/index.html

Appendix A. 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 [2]

Archives of the list can be found at: https://www.ietf.org/mail-archive/web/suit/current/index.html [3]

Authors' Addresses

Brendan Moran
Arm Limited

EMail: Brendan.Moran@arm.com

Hannes Tschofenig
Arm Limited

EMail: hannes.tschofenig@gmx.net

Henk Birkholz
Fraunhofer SIT

EMail: henk.birkholz@sit.fraunhofer.de