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.

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

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

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

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

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

This Internet-Draft will expire on September 6, 2018.
Copyright Notice

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

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

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

Table of Contents

1. Introduction ................................................. 4
2. Conventions and Terminology ................................. 4
3. Requirements .................................................. 5
   3.1. Agnostic to how firmware images are distributed ..... 6
   3.2. Friendly to broadcast delivery ......................... 6
   3.3. Uses state-of-the-art security mechanisms .......... 6
   3.4. Rollback attacks must be prevented ................. 6
   3.5. High reliability ....................................... 6
   3.6. Operates with a small bootloader .................... 7
   3.7. Small Parsers .......................................... 7
   3.8. Minimal impact on existing firmware formats ....... 7
   3.9. Robust permissions .................................. 7
4. Claims ......................................................... 8
5. Architecture .................................................. 9
6. Manifest ....................................................... 11
7. Example Flow .................................................. 12
8. IANA Considerations .......................................... 14
9. Security Considerations ...................................... 14
10. Mailing List Information .................................... 15
11. Acknowledgements ................................. 15
12. References ........................................ 16
  12.1. Normative References ............................ 16
  12.2. Informative References ........................... 16
  12.3. URIs ............................................ 16
Appendix A. Threat Model, User Stories, Security Requirements,
  and Usability Requirements .......................... 17
  A.1. Threat Model ..................................... 17
  A.2. Threat Descriptions ............................... 17
    A.2.1. Threat MFT1: Old Firmware ..................... 17
    A.2.2. Threat MFT2: Mismatched Firmware ............... 18
    A.2.3. Threat MFT3: Offline device + Old Firmware .... 18
    A.2.4. Threat MFT4: The target device misinterprets the type
           of payload ................................... 18
    A.2.5. Threat MFT5: The target device installs the payload
           to the wrong location .......................... 19
    A.2.6. Threat MFT6: Redirection ....................... 19
    A.2.7. Threat MFT7: Payload Verification on Boot ....... 19
    A.2.8. Threat MFT8: Unauthenticated Updates .......... 19
    A.2.9. Threat MFT9: Unexpected Precursor images ...... 20
    A.2.10. Threat MFT10: Unqualified Firmware .......... 20
    A.2.11. Threat MFT11: Reverse Engineering Of Firmware Image
            for Vulnerability Analysis ..................... 21
  A.3. Security Requirements .......................... 21
    A.3.1. Security Requirement MFSR1: Monotonic Sequence
            Numbers ..................................... 21
    A.3.2. Security Requirement MFSR2: Vendor, Device-type
            Identifiers .................................... 22
    A.3.4. Security Requirement MFSR4: Signed Payload Descriptor
            ........................................... 22
    A.3.5. Security Requirement MFSR5: Cryptographic
            Authenticity ................................... 23
            Authenticity ................................... 23
    A.3.7. Security Requirement MFSR7: Firmware encryption ... 23
  A.4. User Stories .................................... 23
    A.4.1. Use Case MFUC1: Installation Instructions ....... 24
    A.4.2. Use Case MFUC2: Reuse Local Infrastructure ..... 24
    A.4.3. Use Case MFUC3: Modular Update ................. 24
    A.4.4. Use Case MFUC4: Multiple Authorisations ....... 25
    A.4.5. Use Case MFUC5: Multiple Payload Formats ...... 25
    A.4.6. Use Case MFUC6: IP Protection .................. 25
  A.5. Usability Requirements .......................... 25
    A.5.1. Usability Requirement MFUR1 .................... 25
    A.5.2. Usability Requirement MFUR2 .................... 25
    A.5.3. Usability Requirement MFUR3 .................... 25
    A.5.4. Usability Requirement MFUR4 .................... 26
    A.5.5. Usability Requirement MFUR5 .................... 26
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 reconfigure the device.

The firmware update process has to ensure that

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

2. Conventions and Terminology

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

This document uses the following terms:
- **Manifest**: The manifest contains meta-data about the firmware image. The manifest is protected against modification and provides information about the author.

- **Firmware Image**: The firmware image is a binary that may contain the complete software of a device or a subset of it. The firmware image may consist of multiple images, if the device contains more than one microcontroller. The image may consist of a differential update for performance reasons. Firmware is the more universal term. Both terms are used in this document and are interchangeable.

The following entities are used:

- **Author**: The author is the entity that creates the firmware image, signs and/or encrypts it and attaches a manifest to it. The author is most likely a developer using a set of tools.

- **Device**: The device is the recipient of the firmware image and the manifest. The goal is to update the firmware of the device.

- **Untrusted Storage**: Firmware images and manifests are stored on untrusted file servers or cloud storage infrastructure. Some deployments may require storage of the firmware images/manifests to be stored on various entities before they reach the device.

3. **Requirements**

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

- Agnostic to how firmware images are distributed
- Friendly to broadcast delivery
- Uses state-of-the-art security mechanisms
- Rollback attacks must be prevented.
- High reliability
- Operates with a small bootloader
- Small Parsers
- 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 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
location for firmware. An alternative approach is to use a 2nd stage bootloader with build-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.

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.

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.

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.

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.

---

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.
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.

There are different types of delivery modes, which are illustrates 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.

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

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

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

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

The manifest format is described in a companion document.

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.

```
+--------+    +-----------------+      +------+
| 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
<-----------------
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 [RFC8240]. This document (and associated specifications) offer a standardized firmware manifest format providing end-to-end security from the author to the device.

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

- installing firmware updates in a robust fashion so that the update does not break the device functionality of the environment this device operates in.

- installing firmware updates in a timely fashion considering the complexity of the decision making process of updating devices, potential re-certification requirements, and the need for user’s consent to install updates.

- the distribution of the actual firmware update, potentially in an efficient manner to a large number of devices without human involvement.

- energy efficiency and battery lifetime considerations.
key management required for verifying the digital signature protecting the manifest.

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

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

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
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


12.2. Informative References


12.3. URIs

[1] mailto:suit@ietf.org
Appendix A. Threat Model, User Stories, Security Requirements, and Usability Requirements

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:

- 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

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
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.

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.

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

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

Mitigates: Threat MFT1 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


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 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)
- 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 digital signature. The authenticity of the manifest can be verified with a digital signature, the authenticity of the firmware image is tied to the manifest by the use of a fingerprint of the firmware image.

Mitigates: Threat MFT8 Implemented by: Signature


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

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

Mitigates: MFT10 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 validated 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 a OEM or Operator of devices, I want to be able to send multiple payload formats to suit the needs of my update, so that I can optimise the bandwidth used by my devices.

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.

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.


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.


A.6.3. Manifest Field: Class ID Condition

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


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.

Implements: Security Requirement MFSR3
A.6.6. Manifest Field: Payload Format

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

Implements: Security Requirement MFSR4, Usability Requirement MFUR5

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 parsing a manifest tree and treat any aliases as top-ranked URIs for the corresponding digest.

Implements: Usability Requirement MFUR2


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 example 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.


Authors’ Addresses

Brendan Moran
Arm Limited

EMail: Brendan.Moran@arm.com

Milosch Meriac
Arm Limited

EMail: Milosch.Meriac@arm.com
Abstract

This specification describes the serialization format of a manifest.

A manifest is a bundle of metadata about the firmware for an IoT device, where to find the firmware, the devices to which it applies, and cryptographic information protecting the manifest.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

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

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

This Internet-Draft will expire on January 3, 2019.

Copyright Notice

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

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

A firmware update mechanism is an essential security feature for IoT devices to deal with vulnerabilities. While the transport of firmware images to the devices themselves is important there are already various techniques available. Equally important is the inclusion of meta-data about the conveyed firmware image (in the form of a manifest) and the use of end-to-end security protection to detect modifications and (optionally) to make reverse engineering more difficult. End-to-end security allows the author, who builds the firmware image, to be sure that no other party (including potential adversaries) is able to install firmware updates on IoT devices without adequate privileges. This authorization process is ensured by the use of dedicated credentials and authorization permissions installed on the IoT device.

This document is part of larger document set: the architecture document can be found in [I-D.ietf-suit-architecture] and the information model of the manifest is described in [I-D.ietf-suit-information-model]. This document focuses on the serialization format.
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].

2.1. Manifest Serialization Format

The following CDDL fragment defines the manifest.

Wherever enumerations are used, they are started at 1. This allows detection of several common software errors that are caused by uninitialised variables.

The processing graph is a mechanism that maps from resources to installed firmware. On one side of the graph are the resources. These are the raw content that a device acquires. Resources can be remote (for example, on a server) or local (for example, an already installed firmware). On the other side of the graph are targets. These are the locations that firmware is installed to. In between these two sides are processors. These are the steps that a device takes to translate raw content into firmware that is installed. In the simplest case, this is a direct mapping; the resource is installed into the target directly. In an example complex case, a device must use decryption, decompression, and differential patching to create the final resource. Differential patching requires that the device refer to an already-installed firmware. In this graph, there are two resources, three processors, and one target. In some cases, one resource may be used by multiple processors, such as a compression table. The nodes of the graph are the resources before or after transformation by a processor and the edges of the graph are the processors themselves.

Resources, processors and targets are marked with node identifiers. Resources have an output node. Targets have an input node. Processors have both.

AuthenticatedManifest = [ 
    authenticatedManifest: COSE_Mac / COSE_Sign, 
    text: bstr .cbor textMap
] 

COSE_Mac = any 
COSE_Sign = any 

textKeys = ( 
    uninitialised: 0 / 
    manifestDescription: 1 /
payloadDescription: 2 /
vendorName: 3 /
modelName: 4 /
payloadVersion: 5
)

textMap = { * textKeys / nint => tstr }

Manifest = [
  manifestVersion : 1,
digestInfo : DigestInfo,

  ; textReference is the digest of the associated
textReference : bstr,
nonce : bstr,
sequence : SequenceNumber,
preConditions : [ * PreCondition ],
postConditions : [ * PostCondition ],
directives : [ * Directive ],
resources : [ * ResourceInfo ],
processors : [ * ProcessingStep ],
targets : [ * TargetInfo ],
extensions : { * int => bstr }
]

ResourceInfo = [
  type: payload:1 / dependency:2 / key:3 / alias:4
indicator: UriList, ; where to find the resource
size: uint / nil, ; size of the resource
        ; (nil when alias)
digest: bstr, ; digest of the resource
onode bstr ; Node of the processing
            ; graph that the resource feeds
]

Processor = [
  decrypt: 1 / decompress: 2 / undiff: 3 /
relocate: 4 / unrelocate: 5,
parameters: bstr ; TBD: more detail needed
inode: bstr, ; Node of the processing graph
        ; that this processor consumes
onode: bstr ; Node of the processing graph
            ; that this processor feeds
]

Target = [
  componentIdentifier: [ * bstr],
storageIdentifier: tstr, ; where to store the resource
encoding: bstr / nil, ; the format of the resource
            ; (nil when alias)
inode: bstr ; Node of the processing graph
            ; that this target consumes

PreCondition = IdCondition / TimeCondition / ImageCondition / CustomCondition
PostCondition = ImageCondition / CustomCondition
IdCondition = [vendor: 1 / class: 2 / device: 3, id: Uuid]
TimeCondition = [installAfter: 4 / bestBefore: 5, time: Timestamp]
CustomCondition = [nint, parameters: bstr]
Directive = [ int => bstr ]
SequenceNumber = uint
Timestamp = uint .size 8
Uuid = bstr .size 16
StorageIdentifier = bstr
ComponentIdentifier = bstr
UriList = { + int => tstr }
DigestInfo = [
            digestAlgorithm : uint,
            ? digestParameters : bstr
            ]

3. IANA Considerations

TBD: Several registries will be required for:
* Standard Conditions *
* Standard Directives *
* Standard Processors *
* Standard text values

4. Security Considerations

This document is about a manifest format describing and protecting firmware images and as such it is part of a larger solution for offering a standardized way of delivering firmware updates to IoT devices. A more detailed discussion about security can be found in the architecture document [I-D.ietf-suit-architecture] and in the information model document [I-D.ietf-suit-information-model].

5. 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

6. Acknowledgements

We would like the following persons for their support in designing this mechanism

- Geraint Luff
- Amyas Phillips
- Dan Ros
- Carsten Bormann
- David Brown
- Markus Gueller
- Frank Audun Kvamtro
- Oyvind Ronningstad

7. References

7.1. Normative References

[I-D.ietf-suit-architecture]

[I-D.ietf-suit-information-model]


Moran & Tschofenig Expires January 3, 2019 [Page 6]
7.2. URIs

[1] mailto:suit@ietf.org

Authors' Addresses

Brendan Moran
Arm Limited

EMail: Brendan.Moran@arm.com

Hannes Tschofenig
Arm Limited

EMail: hannes.tschofenig@gmx.net
Abstract

The IETF SUIT effort has been form to define a secure firmware upgrade solution for Internet of Things (IOT). Recent vulnerabilities and the need to upgrade firmware on the IoT devices for security updates in a standardized, secure, and automated fashion has been the driving force behind this work.

This specification is a requirements document to aid in developing a solution for Secure Firmware upgrade of the IoT devices.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

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

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

This Internet-Draft will expire on May 3, 2018.

Copyright Notice

Copyright (c) 2017 IETF Trust and the persons identified as the document authors. All rights reserved.

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

Table of Contents

1. Introduction . . . . . . . . . . . . . . . . . . . . . . . .   2
2. Solution Requirements . . . . . . . . . . . . . . . . . . . .   2
3. IANA Consideration . . . . . . . . . . . . . . . . . . . . . .   4
4. Security Considerations . . . . . . . . . . . . . . . . . . . .   4
5. Acknowledgements . . . . . . . . . . . . . . . . . . . . . .   4
Authors’ Addresses . . . . . . . . . . . . . . . . . . . . . . .   4

1. Introduction

This draft outlines a set of requirements around firmware download for IoT devices. A sketch of a proposed solution can be found in.

2. Solution Requirements

Informally, a secure firmware upgrade solution might need to address following components:

- Secure firmware description container format, in the form of Manifest
- Locating a server to download the firmware from
- Downloading the manifest and the firmware image(s)
- Cryptographic validation of the manifest and signed code images
- Complete the installation

Given above tasks, this specification breaks down the secure firmware upgrade solution into following requirements:

1. Solution must allow devices that delete the old firmware before installing the new firmware. Thus implying a solution that can easily be implementable on a minimal boot-loader

2. Solution must enable devices that have enough memory to have the new firmware image of the firmware simultaneously loaded with the existing image.

3. The manifest format should be self describing.
4. Allow a given device to decide which manifest format is appropriate for it choosing from JSON, CBOR, or perhaps ASN.1 if there is a a device vendor that plans to use this

5. Manifest must allow metadata about the firmware sourced by a single manufacturer

6. Optionally, the solution may allow the manifest to describe metadata about firmwares from different providers

7. The solution should enable firmware that is delivered as a single image

8. Optionally, the solution may enable firmware to be split into multiple images.

9. The charter should recommend a solution agnostic to the format of the firmware image and inter dependencies. Dependency management is complicated and is by nature proprietary and should not be in the initial scope.

10. The proposed solution must provide mechanism to discover where to download the firmware where that mechanism includes the ability for a local cache.

11. The proposed solution should allow flexibility to choose the underlying transport protocol as defined by the deployment scenarios. The WG should define a MTI set of protocols that firmware servers need to implement and clients can choose which one to use

12. The proposed solution must require a device to validate signatures on the manifest and firmware image(s)

13. Optionally, the solution might want to support encrypted manifest and firmware

14. The proposed solution should enable crypto agility and prevent roll-back attacks.

15. Solution should allow for secure transition between the generations of the keying material

16. Charter should not invent new crypto or transports and use existing techniques
3. IANA Consideration
   Not Applicable

4. Security Considerations
   Not Applicable

5. Acknowledgements
   Thanks IOTSU workshop.

Authors’ Addresses

Suhas Nandakumar
Cisco
Email: snandaku@cisco.com

Cullen Jennings
Cisco
Email: fluffy@iii.ca

Shaun Cooley
Cisco
Email: scooley@cisco.com
Solution Architecture - Secure Firmware Upgrade (SecFU)
draft-nandakumar-suit-secfu-solution-arch-00

Abstract

This specification defines a solution architecture for performing secure firmware upgrade for Internet of Things (IoT). The ulterior motive is to have a framework that is simple, secure, and that uses most common formats and standards in the industry and that works over Internet.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

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

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

This Internet-Draft will expire on May 3, 2018.

Copyright Notice

Copyright (c) 2017 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust’s Legal Provisions Relating to IETF Documents (http://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of

Nandakumar, et al. Expires May 3, 2018
the Trust Legal Provisions and are provided without warranty as
described in the Simplified BSD License.

Table of Contents

1. Introduction ........................................... 2
2. Terminology ........................................... 2
3. Device Considerations ............................... 3
4. Solution Overview .................................. 3
5. Solution Components ................................. 4
   5.1. Manifest ........................................ 4
   5.2. Manifest Format ................................ 6
   5.3. Manifest Security .............................. 6
   5.4. Manifest Optional Extensions ............... 6
   5.5. Firmware Server Discovery .................. 6
   5.6. Firmware Download protocol ................ 7
       5.6.1. Validation Procedures .................. 7
       5.6.2. Manifest Download ...................... 8
       5.6.3. Firmware Download ...................... 8
6. IANA Consideration .................................. 8
7. Security Considerations ............................ 8
8. Acknowledgements ................................... 8
9. Normative References ............................... 9
Authors’ Addresses ..................................... 9

1. Introduction

Internet of Things (IOT) represents a plethora of devices that come in varying flavors of constrained sizes, computing power, and operating considerations. These devices usually need minimal or no management for their operation.

Vulnerabilities within IOT devices have raised serious concerns. There needs to be a way to install or update the firmware on these devices in an automated and secure fashion. A common challenge with the existing firmware update mechanism is they do not work in an automated manner in many environments where IoT devices are deployed. Hence, there is a need to define a firmware update solution that is lightweight, secure, can operate in variety of deployment environments, and is built on well established standards.

2. Terminology

In this document, the key words "MUST", "MUST NOT", "SHOULD", "SHOULD NOT", "MAY", and "OPTIONAL" are to be interpreted as described in RFC 2119 [RFC2119] and indicate requirement levels for compliant implementations.
3. Device Considerations

This draft targets devices that have a boot loader that run in less than 100K bytes of flash and less than 32K bytes of RAM.

There are certain types of devices that delete the firmware image, except for the boot-loader, before proceeding with the upgrade. Alternatively, many devices have sufficient storage to completely download a new firmware image before updating. This solution should be naturally applicable to both.

4. Solution Overview

draft-nandakumar-suit-secfu-requirements captures various requirements that drives the solution defined in this specification.

Below is a high-level solution flow for a successful firmware update on a IoT device.
Successful Firmware Update Flow

on-ready to update firmware

Can access firmware server in the local domain?

<table>
<thead>
<tr>
<th>Yes</th>
<th></th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Download signed manifest from local server well-known URL</td>
<td>Download signed manifest from manufacturer’s pre-configured URL</td>
</tr>
<tr>
<td></td>
<td>Validate manifest via pre-installed public key</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Download the firmware image from the location in manifest</td>
<td></td>
</tr>
<tr>
<td></td>
<td>verify commit hashes on the firmware image</td>
<td></td>
</tr>
<tr>
<td></td>
<td>complete installation</td>
<td></td>
</tr>
</tbody>
</table>

5. Solution Components

Following several sub-sections define various components that makes up the proposed solution architecture

5.1. Manifest

A firmware manifest serves as information representation for metadata about the firmware. A manifest file identifies information about the actual firmware image, its location, applicable device(s), and so on. It is cryptographically signed by the provider (usually the manufacturer) of the firmware.
Minimal Manifest in JSON format

{
  "manifestVersion": "1.0",
  "timestamp": "2017-12-10T15:12:15Z",
  "manufacturer": "manufacturer.com",
  "model": "c7960",
  "firmwareVersion": "10.4.12",
  "firmwareLocation": "well-known location",
  "firmwareCryptoInfo": {
    "commitHash": [
      {
        "digestAlgo": "sha256",
        "hash": "....................."
      },
      {
        "digestAlgo": "sha512",
        "hash": "....................."
      }
    ]
  }
}

"key-info": <most recent public key info>

Above shows example of a minimally defined manifest that identifies the mandatory attributes as explained below

- manifestVersion: Version of the manifest
- timestamp: Time when the manifest was created.
- manufacturer: An identifier of the manufacturer providing the firmware image, represented as String
- model: Device Model, a String
- firmwareVersion: Firmware Version in the format "major.minor.revision"
- firmwareLocation: Location of the firmware images. This can be an absolute URI or a relative URI that is relative to where the manifest was downloaded from.
- firmwareCryptoInfo: Commit Hash information
5.2. Manifest Format

JSON representation is recommended as the default format for describing the manifest. Optionally, formats such as CBOR (see example section) can be used for the same. If more than one format is used, the IoT device can pick one based on its implementation. The firmware download protocol identifies the right format supported by the IoT device.

5.3. Manifest Security

The Manifest file MUST be cryptographically signed by the private key of the manufacturer or the provider of the firmware. This is to ensure source authenticity and to protect integrity of the manifest and the firmware itself.

JWS is the format recommended to store the signed manifest.

\[
\text{signed\_manifest := JWS(manifest.json)}
\]

If CBOR is used for describing the manifest, COSE is recommended for signing.

Optionally, the proposed solution also recommends hash based signatures (hash-sigs) to sign the manifest.

\[
\text{signed\_manifest := hash\_sigs(manifest.json, private\_key)}
\]

5.4. Manifest Optional Extensions

There may be scenarios where the minimalistic manifest defined above may not capture all the requirements for a given deployment setting. In those circumstances, the manifest can be optionally extended to meet the requirements in a extensional specifications.

5.5. Firmware Server Discovery

When it is time for an IoT device to perform a firmware upgrade, the device performs couple of steps to decide the location to download the needed firmware. A device might need to download the new firmware when it is either booting for the first time after deployment or there is a need to upgrade to a newer firmware.

The server discovery procedure starts with the boot-loader attempting to access a server that is local to the domain in which the device operates. The URL to look for a local server is automatically generated using the DHCP domain name.
For example, if the domain name was example.com, and the device was a Cisco 7960, the HTTP URL might be

http://_firmware.example.com/.wellknown/firmware/cisco.com/c7960/manifest.json

In situations where the IoT device cannot access the Internet (factory/enterprise settings, for example), the aforementioned approach might be the only way for the device to perform any kind of firmware or security updates.

However, if the local server cannot be reached or not deployed (say home environments), the device proceeds to download the manifest and firmware from the firmware server URL pre-configured in the bootloader by the manufacturer of the device. For example

http://_firmware.cisco.com/.wellknown/firmware/cisco.com/c7960/manifest.json

If either of the procedures doesn't work, the IoT device is either unusable or might end up running an old version of the firmware.

5.6. Firmware Download protocol

One can envision two possibilities while downloading the firmware:

- Scenarios where the IoT device downloads firmware directly. This is done in order to minimize number of connections. In this scenario, the firmware image must have a digital signature included within the downloaded firmware. The exact placement of this digital signature (prepended, appended, etc) is up to the device manufacturer, but it MUST provided source and integrity guarantees on the entirety of the firmware image and must be verified by the device prior to upgrade.

- Scenarios where a manifest is retrieved and followed by downloading the actual firmware image.

5.6.1. Validation Procedures

The downloaded manifest and firmware is validated before being used:

- Manifest file signature is validated for source and integrity verification. If encrypted, the manifest is decrypted before proceeding with the firmware download.

- On successful validation of the manifest, the device verifies the commit hashes for component(s) of the firmware downloaded against the ones provided in the "firmwareCryptoInfo" section of the manifest.
5.6.2. Manifest Download

Firmware download protocol enables choosing the approach appropriate to the IoT device for downloading the manifest file.

For example, on performing the "Firmware Server Discovery", if a local server is chosen, the device forms a query URL by constructing an endpoint at ".well-known/manifest/<manufacturer>/<model-no>/manifest.json"

Then a HTTP GET request is sent to that URL. For example

http://_firmware.example.com/.well-known/manifest/cisco.com/c7960/manifest.json

The response would be a JSON result of the manifest file. Similarly, the end-point supporting CBOR parsing can request for the CBOR version of the manifest.

5.6.3. Firmware Download

Once the manifest is downloaded and validated, the device proceeds to download the firmware image from the location identified in the firmware manifest. There might be situations where a firmware image is split into multiple files to imply a functional division of the components. This type of firmware can be used by devices that are memory constrained and thus loading the complete image might not be possible. The manifest file may contain the information to indicate the same.

Above example shows use of HTTP as the communication protocol to talk to the firmware server. If the end-point is capable of doing COAP or other protocols, a similar process as above can be applied to retrieve the manifest and the firmware from a well-known place on the local server.

6. IANA Consideration

TODO

7. Security Considerations

TODO - Talk about roaming IoT Device

8. Acknowledgements
9. Normative References


Authors' Addresses

Suhas Nandakumar
Cisco

Email: snandaku@cisco.com

Cullen Jennings
Cisco

Email: fluffy@iii.ca

Shaun Cooley
Cisco

Email: scooley@cisco.com
A Secure and Automatic Firmware Update Architecture for IoT Devices

draft-zhu-suit-automatic-fu-arch-00

Abstract

Firmware update is one of the key features for all Internet of Things (IoT) Devices. Incentives of firmware update is to fix bugs, upgrade configurations, add new services from constrained devices to connected vehicles. The IETF SUIT working group is focusing on defining a firmware update solution that will be usable on Class 1 devices as well as more capable devices with existing mechanisms. This document provides an firmware update architecture, that aims at helping build a secure, automatic and interoperable IoT firmware update mechanism.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

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

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

This Internet-Draft will expire on September 6, 2018.

Copyright Notice

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

This document is subject to BCP 78 and the IETF Trust’s Legal Provisions Relating to IETF Documents (https://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect
1. Introduction

Firmware update is one of the key features for all Internet of Things (IoT) Devices. Incentives of firmware update is to fix bugs, upgrade configurations, add new services from constained devices to connected vehicles. The IETF SUIT working group is focusing on defining a firmware update solution that will be usable on Class 1 devices as well as more capable devices with existing mechanisms. This document provides an firmware update architecture, that aims at helping build a secure, automatic and interoperable IoT firmware update mechanism.

Considering the security aspects of IoT device firmware update, it is not only the cryptographic mechanisms that used to protect the firmware images and the manifest, or the secure transmission protocols, but also the timeliness. An old-fashioned firmware with vulnerabilities is very harmful for an IoT device, which leaves potential target for a cyber attack.

The 2016 Dyn cyberattack is the most impressive example that why all IoT devices SHOULD implement the firmware update. In October 2016, the Dyn cyberattack took place and this huge DDoS attack affected a large number of the US and EU websites. The direct reason of this attack is that many IoT devices such as printers, IP cameras, residential gateways and baby monitors are infected by Mirai malware,
and the attacker used a list of default username/password to access and control the devices. However, one of the actual reasons is that those infected IoT devices are not updated in time. The new version firmware has a mandatory feature to ask the users to change the default username/password pair when they log in.

In order to provide a firmware update in time, the mechanism SHOULD be automatic. The rest of this document describes an automatic solution for the IoT device firmware update. Section 3 provides the architecture. Section 4 provides some of the requirements.

2. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this specification are to be interpreted as described in [RFC2119].

3. Architecture

In the existing IoT firmware update standards such as OMA DM FUMO and LwM2M Firmware Update, the devices usually play the core role during the update procedure. They download the firmware images from a storage location under the guidance of the device management (DM) server, and then setup a status machine and start processing the firmware when they become "Idle".

However, in most IoT use cases, the "Idle" status itself is really confused and hard to reach. For example, a smart electricity meter works 7*24 after it is plugged into the grid. It always runs under "Working" status, then how to execute a firmware update on this kind of devices when needed? Another example is how to update the ECUs in a connected vehicle. The server can only access the ECUs when the vehicle is started, but the ECUs also start to work. Even if the in-car gateway can download the firmware image, it is hardly for a car to update the firmware immediately. When the car gets into "Idle" status, it is usually shut down as well, no ECUs can do further update process.

Moreover, the DM server usually manages multiple different devices at a large amount. The existing standard does not have the scheduling of manage and update firmwares. The DM server MAY store multiple different firmware images at the same time. The firmware images have different functionalities, thus they SHOULD have different urgency levels when updating. For example, a bug fix update firmware has a higher urgency level than a configuration update firmware.
Therefore, there is a need for the secure automatic firmware update mechanism to address the issues mention above. A brief message flow of secure automatic firmware update is shown in Figure 1. There are generally 3 steps. Most of the cryptographic related steps are shown in Figure 4/[I-D.draft-moran-suit-architecture] and not described again in this document.

![Figure 1: Brief message flow of Firmware Update](image)

(A) The server gets the firmware image and related manifest from the firmware author.

(B) The server checks the manifest to know the firmware image urgency class and determine the update mode of the targeted device.

(C) The device downloads the firmware image and manifest, and then start to process the firmware update.

For a secure automatic firmware update solution, there may be three different update mode to be considered.

1. Client-Initiated Update
2. Server-Initiated Update
3. Negotiated Update

3.1. Client-Initiated Update

The Client-Initiated Update is a mechanism that the client update itself unilaterally.

In this case, the client MAY retrieve the latest firmware image information periodically from the server. Once there is a new
firmware image on the server, the client itself started to download the firmware image and manifest. After successful downloading, the client start to process the firmware update when it goes into idle status.

3.2. Server-Initiated Update

The Server-Initiated Update is a mechanism that the server update the device unilaterally.

This mechanism is suggested especially for the 7*24 working device since it has no idle status. Then even if it has downloaded the firmware images from the server. It is hard for the device itself to determine when to process a firmware update.

The traditional way of updating this kind of device is to ask human intervention. This is not a good idea for large amount of devices and also for the automatic solutions. This has already been proved in the Mirai Cyberattack example.

A better solution for this case is to let the server take charge of the update process. The server determines when to update the devices according to the service status retrieved from the device or a pre-provisioned update strategy. The server pushes the firmware image when the device is accessible and trigger the update process based on the previous determinations.

The device in this Solution is considered as an executor and SHOULD follow the commands from server strictly.

3.3. Negotiated Update

The Negotiated Update is the most common use mechanism for existing firmware update standards for IoT devices. E.g. OMA DM and LwM2M.

The server notifies the client once it gets a firmware image from the author. This notification MAY only contain the existence information of a new firmware image. It may also contain the firmware manifest.

The client then decided if it is OK with this update automatically. If the status is idle the client may generates a OK response to the server to ack the notification. Then the server SHOULD start pushing the firmware image to the client; If the status is not good enough for update, the client then generates a response with a scheduled update time. The server then waited and trigger the update after the scheduled update time expires.
4. Requirements

The firmware update mechanism SHOULD be automatic in order to enhance the availability.

The firmware update mechanism SHOULD consider not only single image update, but also multiple images update use cases.

The firmware manifest SHOULD contain enough information for the server to determine the urgency class of the related firmware image.

The server SHOULD be pre-provisioned some strategies to determine the urgency class of received firmware image.

If the denial of service is unavoidable during the firmware update process, this situation SHOULD be notified to the device.

The update mode for each device SHOULD be determined during the registration process.

5. Conclusion

Timeliness is a key consideration when discussing and defining the IoT firmware update mechanism. Unlike the traditional cryptographic considerations that focus more on the confidentiality and integrity, this automatic firmware update solution focuses more on the availability aspect of security when updating an IoT device.

The automatic firmware update solution provided in this document makes the IoT device update in time to shorten the possible vulnerability exposure time for the attackers. This solution also consider the impact of the update process itself SHOULD not cause a denial of service for the device for the minimum period of time.

6. Security Considerations

The whole document can be seen as security considerations for IoT device firmware update.

7. IANA Considerations

TBD.

8. Acknowledgements

TBD.
9. References

9.1. Normative References


9.2. Informative References

[I-D.draft-moran-suit-architecture]

Author’s Address

Jintao Zhu
Huawei
P.R.China

Email: jintao.zhu@huawei.com