SUIT
Internet-Draft
Intended status: Informational
Expires: December 5, 2018

B. Moran
Arm Limited
M. Meriac
Consultant
H. Tschofenig
Arm Limited

June 03, 2018

A Firmware Update Architecture for Internet of Things Devices
draft-ietf-suit-architecture-00

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 will expire on December 5, 2018.

Copyright Notice

Table of Contents

1. Introduction

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

The firmware update process has to ensure that

- The firmware image is authenticated and attempts to flash a malicious firmware image are prevented.

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

2. Conventions and Terminology

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

This document uses the following terms:

- Manifest: The manifest contains meta-data about the 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.

- Bootloader: A bootloader is a piece of software that is executed once a microcontroller has been reset. It is responsible for deciding whether to boot a firmware image that is present or whether to obtain and verify a new firmware image. Since the bootloader is a security critical component its functionality may be split into separate stages. Such a multi-stage bootloader may offer very basic functionality in the first stage and resides in ROM whereas the second stage may implement more complex functionality and resides in flash memory so that it can be updated in the future (in case bugs have been found). The exact split of components into the different stages, the number of firmware images stored by an IoT device, and the detailed functionality varies throughout different implementations.

The following entities are used:

- Author: The author is the entity that creates the firmware image, signs and/or encrypts 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 may be stored on untrusted fileservers or cloud storage infrastructure. Some deployments may require storage of the firmware images/manifests to be stored on various entities before they reach the device.

3. Requirements

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

- Agnostic to how firmware images are distributed

- Friendly to broadcast delivery

- Use state-of-the-art security mechanisms

- Rollback attacks must be prevented

- High reliability

- Operate with a small bootloader

- Small Parsers

- Minimal impact on existing firmware formats

- Robust permissions

- Diverse modes of operation

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

This architecture does not specify any specific broadcast protocol however, given that broadcast may be desirable for some networks, updates must cause the least disruption possible both in metadata and payload transmission.

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. Use 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 migration to post-quantum secure signatures would require significant effort, therefore, mandatory-to-implement support for post-quantum secure signatures is a goal.

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 [RFC7925]. 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, for example AES Key Wrap [RFC5649], in order to maintain friendliness to Content Distribution Networks, bulk storage, and broadcast protocols.

3.4. Rollback attacks must be prevented

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

3.5. High reliability

A power failure at any time must not cause a failure of the device. A failure to validate any part of an update must not cause a failure of the device. One way to achieve this functionality is to provide a minimum of two storage locations for firmware and one bootable 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. Operate 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 that are required to validate at least one signature or MAC 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. These actors might include the following (this is not a comprehensive list).

* A firmware author
* A device OEM
* A device operator
* A network operator
* A device owner

These actors exert their authority on the device by making claims (as in Section 4).

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 may be programmed to 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 claims.

Alternatively, a device may trust precisely one authority, which does all permission management and coordination. Effectively, the authority allows the device to offload complex permissions calculations for the device.

3.10. Operating modes

There are three broad classifications of update operating modes.

- Self initiated
- Third-party initiated
- Hybrid

Self initiated updates take the form of a proactive IoT device that checks for updates. Third-party initiated updates are triggered by an actor other than the IoT device, be it a server, a peer, or a user. Hybrid updates are those that require agreement from both the target IoT device and another actor.

Third-party initiated updates are important to consider because timing of updates may need to be tightly controlled in some high-reliability environments.

An IoT device goes through several steps in the course of an update, each of which can be self-initiated or third-party initiated, or hybrid. An IoT device may go through the following steps, though this is not a comprehensive list.

- Notification
- Pre-authorisation
- Dependency resolution
- Download
- Installation

The notification step consists informing an IoT device that an update is available. This can be accomplished via polling (self-initiated), push notifications (third-party initiated), or more complex mechanisms.

The pre-authorisation step involves verifying the update authority and making a determination that the device is prepared to initiate the fetching and processing of updates. If the device has all information that is necessary to make this determination, then the pre-authorisation may be self-initiated. However, the device can wait for instruction to begin (third-party initiated). Hybrid approaches are possible as well.

A dependency resolution phase is needed when more than one component can be updated or when a differential update is used. The necessary dependencies must be available prior to installation.

The download step is the process of acquiring a local copy of the payload. When the download is self-initiated, this means that the

IoT device chooses when a download occurs and initiates the download process. When a download is third-party initiated, this means that either the remote service tells the IoT device when to download or that it initiates the transfer directly to the IoT device. For example, a download from an HTTP server is initiated locally. A transfer to a LwM2M Firmware Update resource [LwM2M] is initiated remotely.

Installation is the act of processing the payload into a format that the IoT device can recognise.

Each of these steps may require different permissions expressed in claims and may be implemented in a variety of ways.

4. Claims

When a simple set of permissions fails to encapsulate the rules required for a device to 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.

- Trust the actor with access to the referenced shared secret (MAC).

- 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 [SUIT-IM]. 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/authenticated 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/MACs (one or more) or manifest dependencies (one or more).

- Use the instructions provided by the manifest to install the firmware.

- 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. In Figure 1 a firmware image is created by an author, sent to the device and subsequently installed. 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. End-to-end security mechanisms are used to protect the firmware image. Figure 1 does not show the manifest itself, which provides the meta-data about the firmware image and offers the security protection. It may bundled with the firmware image or travel as a standalone item.

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

Figure 1: End-to-End Security.

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 OR the device verifies the MAC covering the manifest using a MAC algorithm. The device is provisioned with a trust anchor that is used to validate the digital signature or MAC 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 illustrated 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/
             \------------/  +-----------+  \------------/
 +--------+                  |           |                 +--------+
 |        |<.................| Untrusted |<................|        |
 | Device |                  | Storage   |                 | Author |
 |        |                  |           |                 |        |
 +--------+                  +-----------+                 +--------+
```

Figure 2: Manifest with attached firmware.

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

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

Figure 3: Independent retrieval of the firmware image.

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

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 on 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 or message authentication codes (MACs).

The manifest information model is described in [SUIT-IM].

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. The firmware and manifest are stored on the same untrusted storage.

```
   +--------+    +-----------------+      +------+
   | Author |    |Untrusted Storage|      |Device|
   +--------+    +-----------------+      +------+
     |                   |                     |
     | Create Firmware   |                     |
     |---------------    |                     |
     |              |    |                     |
     |<--------------    |                     |
     |                   |                     |
     | Upload Firmware   |                     |
     |------------------>|                     |
     |                   |                     |
     | Create Manifest   |                     |
     |----------------   |                     |
     |               |   |                     |
     |<---------------   |                     |
     |                   |                     |
     | Sign Manifest     |                     |
```

```
     |--------------     |                     |
     |             |     |                     |
     |<-------------     |                     |
     |                   |                     |
     | Upload Manifest   |                     |
     |------------------>|                     |
     |                   |                     |
     |                   |   Query Manifest    |
     |                   |<--------------------|
     |                   |                     |
     |                   |   Send Manifest     |
     |                   |-------------------->|
     |                   |                     |
     |                   |                     | Validate Manifest
     |                   |                     |------------------
     |                   |                     |                 |
     |                   |                     |<-----------------
     |                   |                     |
     |                   |  Request Firmware   |
     |                   |<--------------------|
     |                   |                     |
     |                   | Send Firmware       |
     |                   |-------------------->|
     |                   |                     |
     |                   |                     | Verify Firmware
     |                   |                     |---------------
     |                   |                     |              |
     |                   |                     |<--------------
     |                   |                     |
     |                   |                     | Store Firmware
     |                   |                     |--------------
     |                   |                     |             |
     |                   |                     |<-------------
     |                   |                     |
     |                   |                     | Reboot
     |                   |                     |-------
     |                   |                     |      |
     |                   |                     |<------
     |                   |                     |
     |                   |                     | Bootloader validates
     |                   |                     | Firmware
     |                   |                     |----------------------
     |                   |                     |                     |
     |                   |                     |<---------------------
     |                   |                     |
     |                   |                     | Bootloader activates
     |                   |                     | Firmware
     |                   |                     |----------------------
```

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

Figure 4: Example Flow for a Firmware Upate.

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]. A standardized firmware manifest format providing end-to-end security from the author to the device will be specified in a separate document.

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 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

- Marti Bolivar

- Andrzej Puzdrowski

- Markus Gueller

- Henk Birkholz

- Jintao Zhu

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

12. References

12.1. Normative References

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

[RFC7925] Tschofenig, H., Ed. and T. Fossati, "Transport Layer Security (TLS) / Datagram Transport Layer Security (DTLS) Profiles for the Internet of Things", RFC 7925, DOI 10.17487/RFC7925, July 2016, <https://www.rfc-editor.org/info/rfc7925>.

12.2. Informative References

[LwM2M] OMA, ., "Lightweight Machine to Machine Technical Specification, Version 1.0.2", February 2018, <http://www.openmobilealliance.org/release/LightweightM2M/V1_0_2-20180209-A/OMA-TS-LightweightM2M-V1_0_2-20180209-A.pdf>.

[RFC5649] Housley, R. and M. Dworkin, "Advanced Encryption Standard (AES) Key Wrap with Padding Algorithm", RFC 5649, DOI 10.17487/RFC5649, September 2009, <https://www.rfc-editor.org/info/rfc5649>.

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

[SUIT-IM] Moran, B., Tschofenig, H., Birkholz, H., and J. Jimenez, "Firmware Updates for Internet of Things Devices - An Information Model for Manifests", June 2018.

12.3. URIs

[1] mailto:suit@ietf.org

Authors' Addresses

Brendan Moran
Arm Limited

EMail: Brendan.Moran@arm.com

Milosch Meriac
Consultant

EMail: milosch@meriac.com

Hannes Tschofenig
Arm Limited

EMail: hannes.tschofenig@gmx.net