SUIT
Internet Draft
Intended status: Standards Track
Expires: March 2018

M. Pagel
Microsoft Corp

September 12, 2018

A Binary Manifest Serialization Format
draft-pagel-suit-manifest-00

Abstract

This specification describes the serialization format of a software update manifest that is suitable for low-end devices as it eliminates the need to execute a parser.

A manifest is a metadata structure describing 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), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

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."

The list of current Internet-Drafts can be accessed at http://www.ietf.org/ietf/1id-abstracts.txt

The list of Internet-Draft Shadow Directories can be accessed at http://www.ietf.org/shadow.html

This Internet-Draft will expire on March 12, 2019.

Table of Contents

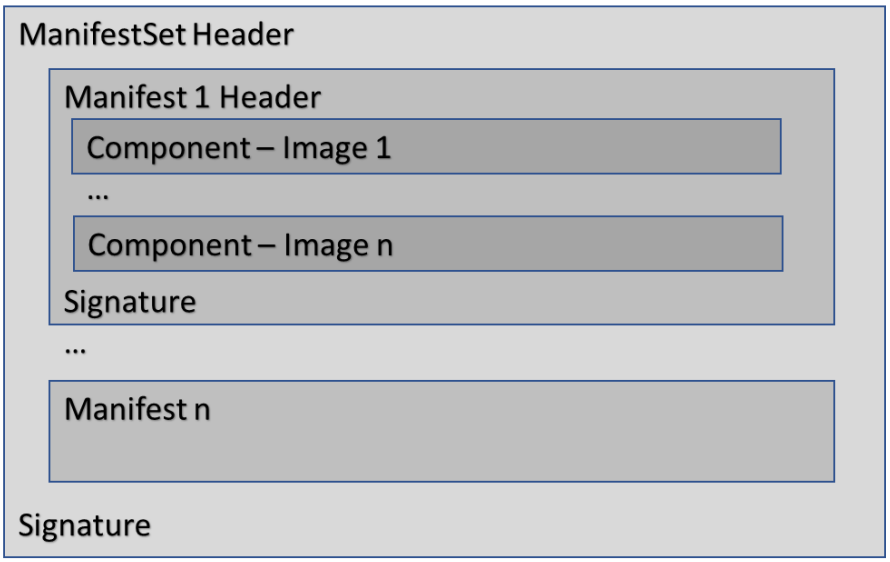
1. Introduction

This document describes a binary format for secured, signed software update “manifests” that is suitable for low-end devices as it eliminates the need to execute a parser.

The SUIT architecture and information model are designed to maximize flexibility. However, in the field we expect each platform provider

to pick a single option to implement within their software stack to keep code as small as possible. For example, basic devices typically support only a single compression or crypto algorithm and associated signature format. Therefore, the manifest used in the field does not need to specify such algorithms as such decision have already been made by the platform provider. SUIT compliant development tools or Update Servers may need to support different options if they want to target multiple device platforms.

We expect each device platform to maintain a set of policies separate from the manifest, which may mandate certain software layers and/or components to be present. The manifest format allows for updating any number of software layers such as drivers, operating systems, and application software. Each layer may consist of multiple software components represented by an image of a particular version of such component. Each such layer may be provided and signed by a different vendor and combined into a manifest set and (in footer) signed by the Network Operator as shown below:



Manifest Structure

Each platform may use a Type id number to identify the type of component and pass such id in the Type parameter to the installer. Each type may also imply a different payload format. The platform may also mandate the order and location each component type gets are installed. A location may be a specific memory partition or separate device such as an SD Card or might even mandate a certain base

memory address. A Flags parameter is provided for a vendor to pass any options, such as location or preprocessing requirements, to the device installer. The platform vendor would need to provide platform specific specifications for the Type and Flags parameters.

To allow platform vendors to support multiple platforms and identify such, it may use the ClassId parameter of the first manifest in a set to identify the platform. Even more importantly, product manufacturers use the ClassId of the last manifest in the set to identify the specific model of product so that the installer can ensure it uses the proper manifest file intended for the product and such model also implies what platform it uses.

To meet privacy requirements, we recommend using transport layer security / channel encryption.

At a bare minimum, a manifest describes a single software image to run. However, manifests might expose richer information, like versioning for application binary interfaces (ABI) or even dependencies between components. These dependencies can be verified before downloading or installing software. For example, an application might depend on a particular version of an operating system. Each component may expose ABIs and consume the ABIs of other components. Each ABI would have a specific ABIType id associated with it. To update components selectively, the manifest specifies a full dependency graph for all components.

The Operator may deliver the latest manifests via broadcast or via an Update Server. The device may call the Update Server with its ClassId and current software configuration. The Update Server may enforce update policies based on such configuration and deliver different manifests accordingly. Policies may include enforcing a certain update sequence, or throttling of installs, or selective test installs, or location specific installs etc.

Rather than including the image URIs in the manifest, the manifest includes only UUID based image descriptors called ImageUid. The device installer receives the manifest and then compares the ImageUids which are currently installed on the device with the ones specified in the manifest and if any have changed, it may request the URIs for those images for download and installation over the network from the Update Server. The Update Server may use a one-time or short-lived URL to limit the availability/distribution of the image. The device may also send its location so that a content distribution network could provide a copy from a nearby file or content cache server, peer device, or in the field via USB thumbdrive. The images may also be received through a broadcast from

other devices. The signature of the manifest guarantees the manifest's authenticity.

2. Pros and Cons vs CBOR based Format

CBOR makes it easier to handle and/or skip optional or new fields whereas a binary structure requires a versioned structure to introduce new fields, which adds complexity to the implementation. However, the binary structure has the advantage that it can be loaded into memory directly without the use of a parser and therefore the installer code is much simpler or smaller. As installers are a common source of bugs and vulnerabilities, simple code is usually considered more secure. It addresses Section 3.6/7 of the architecture document (Small bootloader and parser) quite well. Also, the separation of image URIs allows for a much smaller manifest and therefore reduces memory requirements.

A basic device may not be able to support many options anyways and such devices are more space constrained; the binary format may be a better fit.

A more sophisticated device may offer more options and may use CBOR for other purposes anyways, then the currently proposed format may be more suitable.

3. Manifest Format in Detail

The following tables show the various fields of the manifest set header and signature footer and each manifest with header, image array, and signature footer and the image array with the embedded dependency array. To allow for simple loading, the byte order of numeric fields is considered specific to the platform.

ManifestSetHeader		
Type	Field	Description
UInt32	MagicValue	0x7086760e acting as a static file format signature
UInt16	Version	1 - Version of the manifest set data structure
UInt16	Flags	Hints for device specific policy engine, it can either be interpreted as 16 flags, integer value, or a combination depending on the device
UInt16	ManifestSetDataSize	Size of the total set in bytes

ManifestSetFooter		
Type	Field	Description
UInt8[20]	SignCertThumbprint	Thumbprint of the cert used to sign this manifest. All zeros if the manifest is unsigned.
UInt8[64]	Signature	Digital signature of all the data prior to this field using the signature method specific to the device/platform.

Manifest		
Type	Field	Description
UInt16	Version	Version of the manifest data structure
UInt16	ImageCount	Number of images in the manifest
UInt16	ManifestEntrySize	Size of each entry in bytes, allows safe interpretation even if size changes due to data structure version changes
UInt8[16]	VendorId	UUID5(DNS, "example.com")
UInt8[16]	ClassId	UUID5(VendorId, "Product X")
UInt64	BuildDate	Manifest creation time in unix epoch time
ImageManifestEntry[ImageCount]	ImageEntries	Entries for the images
UInt8[20]	SignCertThumbprint	Thumbprint of the cert used to sign this manifest. All zeros if the manifest is unsigned.
UInt8[64]	Signature	Digital signature of all the data prior to this field using the signature method specific to the device/platform.

ImageManifestEntry		
Type	Field	Description
UInt8[16]	ImageUid	Image UID
UInt8[16]	ComponentUid	UID of the Component the image represents.
UInt16	Type	Component Type (values specific to the device architecture)
UInt32	CompressedImageFileSize	Size of the image file in bytes as compressed
UInt32	UncompressedImageFileSize	Size of the image file in bytes after it is uncompressed
ABIDependency[2]	Provides	Lists any ABI type and version this component provides
ABIDependency[2]	DependsOn	Lists any ABI type and version this component it consumes meaning depends on

ABIDependency		
Type	Field	Description
UInt32	Version	Image UID
UInt32	ABIType	Type of ABI interface

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]. The next few sections address the specific security requirements as defined in the information model:

4.1. MFSR1: Monotonic Sequence Numbers

The BuildDate may be used to enforce sequential updates. However, there are often other methods (e.g., using a hardware root of trust and e-fuses) to block the installation of compromised images.

4.2. MFSR2: Vendor, Device-type Identifiers

The array of ImageUIDs provides the specific set of images which need to be installed on the device.

4.3. MFSR3: Best-Before Timestamps

This requirement appears to be optional. In case you are concerned about this case, an installer could enforce that a manifest is only valid for a particular timeframe from the BuildDate. The Update Server would re-sign (with a new BuildDate) close to the expiry time.

4.4. MFSR5: Cryptographic Authenticity

Each manifest (and each image file) is signed.

4.5. MFSR4a/b: Authenticated Payload Type and Storage Location

Each image has a Type identifier. The device software uses its own policy to determine which image types are supported and which location they are installed. If a component can be installed in various locations, the Flags parameter can be used to specify preferred location.

4.6. MFSR4c: Authenticated Remote Resource Location

Once the manifest is processed and the images to update are identified, the device may request a download location from an Update Server.

4.7. MFSR4d: Secure Boot

We certainly encourage that both the installer and bootloader verify the authenticity of the manifest.

4.8. MFSR4e: Authenticated precursor images

As IoT devices may not be able to connect to the Internet to receive updates for a long period of time, we do not believe that sequential installation is practical and therefore the current proposal does not allow for this option. However, we do believe the proposal contains enough flexibility that support could be added later

4.9. MFSR4f: Authenticated Vendor and Class IDs

Both the Vendor and Class Id are part of the signed manifest body.

4.10. MFSR6: Rights Require Authenticity

Rights management is outside of the scope of the manifest format, but a device or Update Server may enforce them.

4.11. MFSR7: Firmware encryption

A platform may mandate image encryption for any or all components. If encryption is optional, the vendor may need to specify such fact in the Flags parameter.

5. IANA Considerations

TBD

6. 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].

7. 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

8. References

8.1. Normative References

[I-D.ietf-suit-architecture]

Moran, B., Meriac, M., Tschofenig, H., and D. Brown, "A Firmware Update Architecture for Internet of Things Devices", draft-ietf-suit-architecture-01 (work in progress), July 2018.

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

Moran, B., Tschofenig, H., Birkholz, H., and J. Jimenez, "Firmware Updates for Internet of Things Devices - An Information Model for Manifests", draft-ietf-suit-information-model-01 (work in progress), June 2018.

Author's Addresses

Martin Pagel

Microsoft Corp

Email: martin.pagel@microsoft.com