SUIT
Internet-Draft
Intended status: Informational
Expires: 12 January 2023

B. Moran
Arm Limited
H. Birkholz
Fraunhofer SIT

11 July 2022

Secure Reporting of Update Status
draft-ietf-suit-report-02

Abstract

The Software Update for the Internet of Things (SUIT) manifest provides a way for many different update and boot workflows to be described by a common format. However, this does not provide a feedback mechanism for developers in the event that an update or boot fails.

This specification describes a lightweight feedback mechanism that allows a developer in possession of a manifest to reconstruct the decisions made and actions performed by a manifest processor.

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 12 January 2023.

Copyright Notice

Table of Contents

1. Introduction

A SUIT manifest processor can fail to install or boot an update for many reasons. Frequently, the error codes generated by such systems fail to provide developers with enough information to find root causes and produce corrective actions, resulting in extra effort to reproduce failures. Logging the results of each SUIT command can simplify this process.

While it is possible to report the results of SUIT commands through existing logging or attestation mechanisms, this comes with several drawbacks:

* data inflation, particularly when designed for text-based logging

* missing information elements

* missing support for multiple components

The CBOR objects defined in this document allow devices to:

* report a trace of how an update was performed

* report expected vs. actual values for critical checks

* describe the installation of complex multi-component architectures

* describe the measured properties of a system

* report the exact reason for a parsing failure

This document provides a definition of a SUIT-specific logging container that may be used in a variety of scenarios.

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 BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

Terms used in this specification include:

* Boot: initialization of an executable image. Although this specification refers to boot, any boot-specific operations described are equally applicable to starting an executable in an OS context.

3. The SUIT Record

If the developer can be assumed to have a copy of the manifest, then they need little information to reconstruct what the manifest processor has done. They simply need any data that influences the control flow of the manifest. The manifest only supports the following control flow primitives:

* Set Component/Dependency Index

* Set/Override Parameters

* Try-Each

* Run Sequence

* Conditions

Of these, only conditions change the behavior of the processor from the default, and then only when the condition fails.

Then, to reconstruct the flow of a manifest, all a developer needs is a list of metadata about failed conditions:

* the current manifest

* the current section

* the offset into the current section

* the current component index

* the "reason" for failure

Most conditions compare a parameter to an actual value, so the "reason" is typically simply the actual value.

Since it is possible that a non-condition command (directive) may fail in an exceptional circumstance, this must be included as well. However, a failed directive will terminate processing of the manifest. To accommodate for a failed command and for explicit "completion," an additional "result" element is added as well. In the case of a command failure, the failure reason is typically a numeric error code. However, these error codes need to be standardised in order to be useful.

Reconstructing what a device has done in this way is compact, however it requires some reconstruction effort. This is an issue that can be solved by tooling.

```
SUIT_Record = {
    suit-record-manifest-id        => [* uint ],
    suit-record-manifest-section   => int,
    suit-record-section-offset     => uint,
    (
        suit-record-component-index  => uint //
        suit-record-dependency-index => uint
```

```
    ),
    suit-record-properties     => SUIT_Parameters,
}
```

suit-record-manifest-id is used to identify which manifest contains the command that caused the record to be generated. The manifest id is a list of integers that form a walk of the manifest tree, starting at the root. An empty list indicates that the command was contained in the root manifest. If the list is not empty, the command was contained in one of the root manifest's dependencies, or nested even further below that.

For example, suppose that the root manifest has 3 dependencies and each of those dependencies has 2 dependencies of its own:

* Root
  - Dependency A

    - Dependency A0
    - Dependency A1
  - Dependency B
    - Dependency B0
    - Dependency B1
  - Dependency C
    - Dependency C0
    - Dependency C1

A manifest-id of [1,0] would indicate that the current command was contained within Dependency B0. Similarly, a manifest-id of [2,1] would indicate Dependency C1

suit-record-manifest-section indicates which section of the manifest was active. This is used in addition to an offset so that the

developer can index into severable sections in a predictable way. The value of this element is the value of the key that identified the section in the manifest.

suit-record-section-offset is the number of bytes into the current section at which the current command is located.

suit-record-component-index is the index of the component that was specified at the time that the report was generated. This field is necessary due to the availability of set-current-component values of True and a list of components. Both of these values cause the manifest processor to loop over commands using a series of component-ids, so the developer needs to know which was selected when the command executed.

suit-record-dependency-index is similar to suit-record-component-index but is used to identify the dependency that was active.

suit-record-properties contains any measured properties that led to the command failure. For example, this could be the actual value of a SUIT_Digest or class identifier. This is encoded in a SUIT_Parameters block as defined in [I-D.ietf-suit-manifest].

4. The SUIT Report

Some metadata is common to all records, such as the root manifest: the manifest that is the entry-point for the manifest processor. This metadata is aggregated with a list of SUIT_Records. The SUIT_Report may also contain a list of any system properties that were measured and reported, and a reason for a failure if one occured.

```
SUIT_Report = {
  suit-report-manifest-digest => SUIT_Digest,
  ? suit-report-manifest-uri  => tstr,
  ? suit-report-nonce         => bstr,
  suit-report-records         => [ * SUIT_Record ],
  ? suit-system-properties      => [ + system-property-claims ],
```

```
  suit-report-result          => true / {
    suit-report-result-code   => int, ; could condense to enum later
    suit-report-result-record => SUIT_Record,
  }
}
system-property-claims = {
  system-component-id => SUIT_Component_Identifier,
  + SUIT_Parameters,
}
```

suit-report-manifest-digest provides a SUIT_Digest (as defined in [I-D.ietf-suit-manifest]) that is the characteristic digest of the Root manifest.

suit-report-manifest-uri provides the reference URI that was provided in the root manifest.

suit-report-nonce provides a container for freshness or replay protection information. This field MAY be omitted where the suit-report is authenticated within a container that provides freshness already. For example, attestation evidence typically contains a proof of freshness.

suit-system-properties provides a list of measured or asserted properties of the system that creates the suit report. These properties are scoped by component identifier. Because this list is expected to be constructed on the fly by a constrained node, component identifiers may appear more than once. A recipient may convert the result to a more conventional structure:


```
SUIT_Record_System_Properties = {
  * component-id => {
    + SUIT_Parameters,
  }
}
```

suit-report-records is a list of 0 or more SUIT Records. Because SUIT Records are only generated on failure, in simple cases this can

be an empty list.

suit-report-result provides a mechanism to show that the SUIT procedure completed successfully (value is true) or why it failed (value is a map of an error code and a SUIT_Record).

The suit-report-result-code indicates the reason for the failure. Values are expected to be CBOR parsing failures, Schema validation failures, COSE validation failures or SUIT processing failures.

The suit-report-result-record indicates the exact point in the manifest or manifest dependency tree where the error occured.

5. Attestation

This document ~~can allow~~ describes how a well-informed verifier can infer the trustworthiness of a remote device. Remote attestation is done by using the SUIT_Manifest_Envelope along with the SUIT_Report to reconstruct the state of the device at boot time. By embedding data used for remote attestation in the SUIT_Report, a remote device can use an append-only log to collect both measurements and debug/failure information into the same document. This document can then be conveyed to a verifier as a part of the attestation evidence. A remote attestation format to convey attestation evidence, such as an Entity Attestation Token (EAT, see [I-D.ietf-rats-eat]), that contains a SUIT_Report MUST also include an integrity measurement of the Manifest Parser & Report Generator.

When a Concise Reference Integrity Manifest (CoRIM, see [I-D.birkholz-rats-corim] is delivered in a SUIT_Manifest_Envelope, this codifies the delivery of verification information to the verifier:

* The Firmware Distributor:
  - sends the SUIT_Manifest_Envelope to the Verifier without payload or text, but with CoRIM
  - sends the SUIT_Manifest_Envelope to the recipient without CoRIM, or text, but with payload

* The Recipient:

- Installs the firmware as described in the SUIT_Manifest and generates a SUIT_report, which is encapsulated in an EAT by the installer and sent to the Firmware Distributor.

- Boots the firmware as described in the SUIT_Manifest and creates a SUIT_report, which is encapsulated in an EAT by the installer and sent to the Firmware Distributor.

* The Firmware Distributor sends both reports to the verifier (separately or together)

* The Verifier:

  - Reconstructs the state of the device using the manifest

  - Compares this state to the CoRIM

  - Returns an Attestation Report to the Firmware Distributor

This approach simplifies the design of the bootloader since it is able to use an append-only log. It allows a verifier to validate this report against a signed CoRIM that is provided by the firmware author, which simplifies the delivery chain of verification information to the verifier.

This information is not intended as Attestation Evidence and while an Attestation Report MAY provide this information for conveying error codes and/or failure reports, it SHOULD be translated into general-purpose claims for use by the Relying Party.

6. IANA Considerations

IANA is requested to allocate a CBOR tag for the SUIT Report.

7. Security Considerations

The SUIT Report should either be carried over a secure transport, or signed, or both. Ideally, attestation should be used to prove that the report was generated by legitimate hardware.

8. Acknowledgements

9. References

9.1. Normative References

[I-D.ietf-suit-manifest]
Moran, B., Tschofenig, H., Birkholz, H., and K. Zandberg, "A Concise Binary Object Representation (CBOR)-based Serialization Format for the Software Updates for Internet of Things (SUIT) Manifest", Work in Progress, Internet-Draft, draft-ietf-suit-manifest-18, 11 July 2022, <https://www.ietf.org/archive/id/draft-ietf-suit-manifest-18.txt>.

[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>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

9.2. Informative References

[I-D.birkholz-rats-corim]
Birkholz, H., Fossati, T., Deshpande, Y., Smith, N., and W. Pan, "Concise Reference Integrity Manifest", Work in Progress, Internet-Draft, draft-birkholz-rats-corim-03, 11 July 2022, <https://www.ietf.org/archive/id/draft-birkholz-rats-corim-03.txt>.

[I-D.ietf-rats-eat]
Lundblade, L., Mandyam, G., and J. O'Donoghue, "The Entity Attestation Token (EAT)", Work in Progress, Internet-Draft, draft-ietf-rats-eat-14, 10 July 2022, <https://www.ietf.org/archive/id/draft-ietf-rats-eat-14.txt>.

Authors' Addresses

Brendan Moran
Arm Limited
Email: Brendan.Moran@arm.com

Henk Birkholz
Fraunhofer SIT
Email: henk.birkholz@sit.fraunhofer.de