RATS
Internet-Draft
Intended status: Standards Track
Expires: January 14, 2021

H. Birkholz
Fraunhofer SIT
B. Moran
Arm Limited

July 13, 2020

Trustworthiness Vectors for the Software Updates of Internet of Things (SUIT) Workflow Model
draft-birkholz-rats-suit-claims-00

Abstract

The IETF Remote Attestation Procedures (RATS) architecture defines Conceptual Messages as input and output of the appraisal process that assesses the trustworthiness of remote peers: Evidence and Attestation Results. Based on the Trustworthiness Vectors defined in Trusted Path Routing, this document defines a core set of Claims to be used in Evidence and Attestation Results for the Software Update for the Internet of Things (SUIT) Workflow Model. Consecutively, this document is in support of the Trusted Execution Environment Provisioning (TEEP) architecture, which defines the assessment of remote peers via RATS and uses SUIT for evidence generation as well as a remediation measure to improve trustworthiness of given remote peers.

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 January 14, 2021.

Table of Contents

1. Introduction

Attestation Results are an essential output of Verifiers as defined in the Remote ATtestation procedureS (RATS) architecture [I-D.ietf-rats-architecture]. They are consumed by Relying Parties: the entities that intend to build future decisions on trustworthiness assessments of remote peers. Attestation Results must be easily

digestable by Relying Parties - in contrast to the rather complex or domain-specific Evidence digested by Verifiers.

In order to create Attestation Results, a Verifier must consume Evidence generated by a given Attester (amongst other Conceptual Messages, such as Endorsements and Attestation Policies). Both Evidence and Attestation Results are composed of Claims. This document highlights and defines a set of Claims to be used in Evidence and Attestation Results that are based on the SUIT Workflow Model [I-D.ietf-suit-manifest]. In the scope of this document, an Attester takes on the role of a SUIT Recipient: the system that receives a SUIT Manifest.

1.1. SUIT Workflow Model and Procedures

This document focuses on Evidence and Attestation Results that can be generated based on the output of SUIT Procedures. The SUIT Workflow Model allows for two types of SUIT Procedures generating Reports on the Attester as defined in the SUIT Manifest specification:

Update Procedures: A procedure that updates a device by fetching dependencies, software images, and installing them

An Update Procedure creates a Report on mutable software components that are installed or updated on hardware components.

Boot Procedures: A procedure that boots a device by checking dependencies and images, loading images, and invoking one or more image

A Boot Procedure creates a Report on measured boot events (e.g. during Secure Boot).

The Records contained in each type of Report can be used as Claims in Evidence generation on the Attester and in Attestation as described in this document. Analogously, a corresponding Verifier appraising that Evidence can create Attestation Results using the Claims defined in this document.

Both types of SUIT Procedures pass several stages (e.g. dependency-checking is one stage). The type and sequence of stages are defined by the Command Sequences included in a SUIT Manifest. For each stage in which a Command from the Command Sequence is executed a Record is created. All Records of a SUIT procedure contain binary results limited to "fail" or "pass". The aggregated sequence of all Records is composed into a Report.

This document specifies new Claims derived from Command Sequence Reports and highlights existing Claims as defined in Trusted Path Routing [I-D.voit-rats-trusted-path-routing] that are applicable to the operational state of installed and updated software.

The Claims defined in this document are in support of the Trusted Execution Environment Provisioning (TEEP) architecture. During TEEP, the current operational state of an Attester is assessed via RATS. If the corresponding Attestation Results - as covered in this document - indicate insufficient Trustworthiness Levels with respect to installed software, the SUIT Workflow Model is used for remediation.

1.2. Terminology

This document uses the terms and concepts defined in [I-D.ietf-rats-architecture], [I-D.ietf-suit-manifest], and [I-D.ietf-teep-architecture].

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.

2. Trustworthiness Vectors

While there are usage scenarios where Attestation Results can be binary decisions, more often than not the assessment of trustworthiness is represented by a more fine-grained spectrum or based on multiple factors. These shades of Attestation Results are captured by the definition of Trustworthiness Vectors in Trusted Path Routing [I-D.voit-rats-trusted-path-routing]. Trustworthiness Vectors are sets of Claims representing appraisal outputs created by a Verifier. Each of these Claims is called a Trustworthiness Level. Multiple Trustworthiness Levels are composed into a vector.

An Attester processing SUIT Manifests can create three types of Claims about its Target Environments. This includes Claims about:

- installed manifests including initial state (e.g. factory default),
- hardware component identifiers that represent the targets of updates, and
- SUIT Interpreter results (e.g. test-failed) created during updates.

Every SUIT Manifest maps to a certain intended state of a device. Every intended device composition of software components associated with hardware components can therefore be expressed based on a SUIT Manifest. The current operational state of a device can be represented in the same form, including the initial state.

As a result, the Claims defined in this document are bundled by the scope of the information represented in SUIT Manifests, i.e., dedicated blobs of software that are the payload of a SUIT Manifest. All Claims associated with an identifiable SUIT Manifest must always be bundled together in a Claims set that is limited to the Claims defined in this document.

3. SUIT Claims

The Claim description in this document uses CDDL as the formal modeling language for Claims. This approach is derived from [I-D.ietf-rats-eat]. All Claims are based on information elements as used in the SUIT Manifest specification [I-D.ietf-suit-manifest]. For instance, a SUIT Vendor ID is represented as an UUID. Analogously, the corresponding vendor-identifier Claim found below is based on a UUID. SUIT Claims are differentiated in:

- software and hardware characteristics (System Properties), and

- reports about updates their SUIT Commands (SUIT Records).

Both types of Claims are always bundled in dedicated Claim Sets. Implementations can encode this information in various different ways (data models), e.g., sets, sequences, or nested structures. The following subsections define the SUIT Report Claims for RATS and are structured according to the following CDDL expression.

```
suit-report = {
  suit-system-properties => [ + system-property-claims ],
  suit-records => [ + interpreter-record-claims ],
}

system-property-claims => { + $$system-property-claim }
interpreter-record-claims => { + $$interpreter-record-claim }
```

3.1. System Properties Claims

System Properties Claims are composed of:

- Hardware Component Claims and

- Software Component Claims.

Correspondingly, the Claim definitions below highlight if a Claim is generic or hw/sw-component specific.

3.1.1. vendor-identifier

A RFC 4122 UUID representing the vendor of the Attester or one of its hardware and/or software components.

$$system-property-claim //= (vendor-identifier => RFC4122_UUID)

3.1.2. class-identifier

A RFC 4122 UUID representing the class of the Attester or one of its hardware and/or software components.

$$system-property-claim //= (class-identifier => RFC4122_UUID)

3.1.3. device-identifier

A RFC 4122 UUID representing the Attester.

$$system-property-claim //= (device-identifier => RFC4122_UUID)

3.1.4. component-identifier

A sequence of binary identifiers that is intended to identify a software-component of an Attester uniquely. A binary identifier can represent a CoSWID [I-D.ietf-sacm-coswid] tag-id.

$$system-property-claim //= (class-identifier => [+ identifier])

3.1.5. image-digest

A fingerprint computed over a software component image on the Attester. This Claim is always bundled with a component-identifier or component-index.

$$system-property-claim //= (image-digest => digest)

3.1.6. image-size

The size of a firmware image on the Attester.

$$system-property-claim //= (image-size => size)

3.1.7. minimum-battery

The configured minimum battery level of the Attester in mWh.

$$system-property-claim //= (minimum-battery => charge)

3.1.8. version

The Version of a hardware or software component of the Attester.

$$system-property-claim //= (version => version-value)

3.2. Interpreter Record Claims

This class of Claims represents the content of SUIT Records generated by Interpreters running on Recipients. They are always bundled into Claim Sets representing SUIT Reports and are intended to be included in Evidence generated by an Attester. The Interpreter Record Claims appraised by a Verifier can steer a corresponding a Firmware Appraisal procedures that consumes this Evidence. Analogously, these Claims can be re-used in generated Attestation Results as Trustworthiness Vectors [I-D.voit-rats-trusted-path-routing].

3.2.1. record-success

The result of a Command that was executed by the Interpreter on an Attester.

$$interpreter-record-claim //= (record-success => bool)

3.2.2. component-index

A positive integer representing an entry in a flat list of indices mapped to software component identifiers to be updated.

$$system-property-claim //= (component-index => uint)

3.2.3. dependency-index

A thumbprint of a software component that an update depends on.

$$interpreter-record-claim //= (dependency-index => digest)

3.2.4. command-index

A positive integer representing an entry in a SUIT_Command_Sequence identifying a Command encoded as a SUIT Manifest Directive or SUIT Manifest Condition.

```
$$interpreter-record-claim //= ( command-index => uint )
```

3.2.5. nominal-parameters

A list of SUIT_Parameters associated with a specific Command encoded as a SUIT Manifest Directive.

```
$$interpreter-record-claim //= ( nominal-parameters => parameter-list )
```

3.2.6. nominal-parameters

A list of SUIT_Parameters associated with a specific Command that was executed by the Interpreter on an Attester.

```
$$interpreter-record-claim //= ( actual-parameters => parameter-list )
```

3.3. Generic Record Conditions (TBD)

- test-failed
- unsupported-command
- unsupported-parameter
- unsupported-component-id
- payload-unavailable
- dependency-unavailable
- critical-application-failure
- watchdog-timeout

4. List of Commands (TBD)

- Check Vendor Identifier
- Check Class Identifier
- Verify Image
- Set Component Index
- Override Parameters
- Set Dependency Index

- Set Parameters
- Process Dependency
- Run
- Fetch
- Use Before
- Check Component Offset
- Check Device Identifier
- Check Image Not Match
- Check Minimum Battery
- Check Update Authorized
- Check Version
- Abort
- Try Each
- Copy
- Swap
- Wait For Event
- Run Sequence
- Run with Arguments

5. References

5.1. 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>.

[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>.

5.2. Informative References

[I-D.ietf-rats-architecture]
Birkholz, H., Thaler, D., Richardson, M., Smith, N., and W. Pan, "Remote Attestation Procedures Architecture", draft-ietf-rats-architecture-05 (work in progress), July 2020.

[I-D.ietf-rats-eat]
Mandyam, G., Lundblade, L., Ballesteros, M., and J. O'Donoghue, "The Entity Attestation Token (EAT)", draft-ietf-rats-eat-03 (work in progress), February 2020.

[I-D.ietf-sacm-coswid]
Birkholz, H., Fitzgerald-McKay, J., Schmidt, C., and D. Waltermire, "Concise Software Identification Tags", draft-ietf-sacm-coswid-15 (work in progress), May 2020.

[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", draft-ietf-suit-manifest-08 (work in progress), July 2020.

[I-D.ietf-teep-architecture]
Pei, M., Tschofenig, H., Thaler, D., and D. Wheeler, "Trusted Execution Environment Provisioning (TEEP) Architecture", draft-ietf-teep-architecture-11 (work in progress), July 2020.

[I-D.voit-rats-trusted-path-routing]
Voit, E., "Trusted Path Routing", draft-voit-rats-trusted-path-routing-02 (work in progress), June 2020.

Authors' Addresses

Henk Birkholz
Fraunhofer SIT

EMail: henk.birkholz@sit.fraunhofer.de

Brendan Moran
Arm Limited

EMail: Brendan.Moran@arm.com