SUIT
Internet-Draft
Intended status: Standards Track
Expires: 29 May 2026

B. Moran
Arm Limited
H. Birkholz
Fraunhofer SIT

25 November 2025

Secure Reporting of SUIT Update Status
draft-ietf-suit-report-17

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. This document specifies 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 29 May 2026.

Table of Contents

1. Introduction 2
2. Conventions and Terminology 3
3. The SUIT_Record 4
4. The SUIT_Report 7
4.1. suit-report-records 8
4.2. SUIT_Report Result 9
5. Attestation 10
6. Capability Reporting 13
7. EAT Claim 14
8. SUIT_Report Container 14
9. IANA Considerations 15
9.1. Expert Review Instructions 16
9.2. Media Type Registration 17
9.2.1. application/suit-report+cose 17
9.3. CoAP Content-Format Registration 18
9.4. CBOR Tag Registration 18
9.5. SUIT_Report Elements 18
9.6. SUIT_Record Elements 19
9.7. SUIT_Report Reasons 20
9.8. SUIT Capability Report Elements 21
10. Security Considerations 23
11. Acknowledgements 24
12. References 24
12.1. Normative References 24
12.2. Informative References 26
Appendix A. Full CDDL 26
Authors' Addresses 30

1. Introduction

This document specifies a logging container, specific to Software Update for the Internet of Things (SUIT) Manifests ([I-D.ietf-suit-manifest]) that creates a lightweight feedback mechanism for developers in the event that an update or boot fails in the manifest processor. In this way, it provides the necessary link between the Status Tracker Client and the Status Tracker Server as defined in Section 2.3 of [RFC9019].

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

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.

The terms "Author", "Recipient", and "Manifest" are defined in Section 2 of [I-D.ietf-suit-manifest].

Additionally, this document uses the term Boot: initialization of an executable image. Although this document refers to boot, any boot-specific operations described are equally applicable to starting an executable in an OS context.

3. The SUIT_Record

The SUIT_Record is a record of a decision taken by the Manifest Processor. It contains the information that the Manifest Processor used to make the decision. The decision can be inferred from this information, so it is not included. If the developer has a copy of the manifest, then they need little information to reconstruct what the manifest processor has done. They need any data that influences the control flow of the manifest. The manifest only supports the following control flow primitives:

* Set Component
* 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.

To reconstruct the flow of a manifest, a developer needs a list of metadata about failed conditions:

* the current manifest
* the current Command Sequence (Section 5.3.3 of [I-D.ietf-suit-manifest])
* the offset into the current Command Sequence
* the current component index
* the "reason" for failure

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

Since it is possible that a non-condition command (directive) may fail in an exceptional circumstance, a failure code for a non condition command must be communicated to the developer. 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 included as well; however, this is included in the SUIT_Report (Section 4). 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.

This approach effectively compacts the log of operations taken using the SUIT Manifest as a dictionary. This enables a full reconstruction of the log using a matching decompaction tool. The following CDDL ([RFC8610]) shows the structure of a SUIT_Record.

```
SUIT_Record = [
    suit-record-manifest-id        : [* uint ],
    suit-record-manifest-section   : int,
    suit-record-section-offset     : uint,
    suit-record-component-index    : uint,
    suit-record-properties         : SUIT_Parameters,
    $$SUIT_Record_Extensions
]
```

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 (index 0)
    - Dependency AA (index 0,0)
    - Dependency AB (index 0,1)
  - Dependency B (index 1)
    - Dependency BA (index 1,0)
    - Dependency BB (index 1,1)
  - Dependency C (index 2)
    - Dependency CA (index 2,0)
    - Dependency CB (index 2,1)

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

suit-record-manifest-section indicates which Command Sequence of the manifest was active. Only the "top level" Command Sequences, with entries in the Manifest are identified by this element. These are:

* suit-validate = 7

* suit-load = 8

* suit-invoke = 9

* suit-dependency-resolution = 15

* suit-payload-fetch = 16

* suit-candidate-verification = 18

* suit-install = 20

This list may be extended through extensions to the SUIT_Manifest.

suit-record-manifest-section is used in addition to an offset so that the developer can index into severable Command Sequences in a predictable way. The value of this element is the value of the key that identified the Command Sequence in the manifest.

suit-record-section-offset is the number of bytes into the current Command Sequence 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-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 Section 8.4.8 of [I-D.ietf-suit-manifest].

4. The SUIT_Report

The SUIT_Report is a SUIT-specific logging container. It contains the SUIT_Records needed to reconstruct the decisions made by a Manifest Processor as well as references to the Manifest being processed, the result of processing, and an optional capability 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 as defined in Section 3. 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 occurred. The following CDDL describes the structure of a SUIT_Report and a SUIT_Reference:

```
SUIT_Report = {
  suit-reference              => SUIT_Reference,
  ? suit-report-nonce         => bstr,
  suit-report-records         => \
        \[ * SUIT_Record / system-property-claims \],
  suit-report-result          => true / {
    suit-report-result-code   => int,
    suit-report-result-record => SUIT_Record,
    suit-report-result-reason => SUIT_Report_Reasons,
  },
  ? suit-report-capability-report => SUIT_Capability_Report,
  $$SUIT_Report_Extensions
}

system-property-claims = {
  system-component-id => SUIT_Component_Identifier,
  + SUIT_Parameters,
}

SUIT_Reference = [
    suit-report-manifest-uri  : tstr,
    suit-report-manifest-digest : SUIT_Digest,
]
```

Further details for each element appear in subsequent sections: the encoding of suit-report-records is defined in Section 4.1, the result semantics in Section 4.2, and the optional capability report in Section 6.

The suit-reference provides a reference URI and digest for a suit manifest. The URI MUST exactly match the suit-reference-uri (Section 8.4.3 of [I-D.ietf-suit-manifest]) that is provided in the

manifest. The digest is the digest of the manifest, exactly as reported in SUIT_Authentication, element 0 (Section 8.3 of [I-D.ietf-suit-manifest]).

NOTE: The digest is used in preference to other identifiers in the manifest because it allows a manifest to be uniquely identified (collision resistance) whereas other identifiers, such as the sequence number, can collide, particularly in scenarios with multiple trusted signers.

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-report-manifest-digest provides a SUIT_Digest (as defined in Section 10 of [I-D.ietf-suit-manifest]) that is the characteristic digest of the Root manifest. This digest MUST be the same digest as is held in the first element of SUIT_Authentication in the referenced Manifest_Envelope.

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

4.1. suit-report-records

suit-report-records is a list of 0 or more SUIT_Records or system-property-claims. Because SUIT_Records are only generated on failure, in simple cases this can be an empty list. SUIT_Records and suit-system-property-claims are merged into a single list because this reduces the overhead for a constrained node that generates this report. The use of a single log allows report generators to use simple memory management. Because the system-property-claims are encoded as maps and SUIT_Records are encoded as lists, a recipient need only filter the CBOR Type-5 entries from suit-report-records to obtain all system-property-claims.

System Properties can be extracted from suit-report-records by filtering suit-report-records for maps. System Properties are 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,
  }
}
```

4.2. SUIT_Report Result

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).

suit-report-result-reason gives a high-level explanation of the failure. These reasons are intended for interoperable implementations. The reasons are divided into a small number of groups:

* suit-report-reason-cbor-parse: a parsing error was encountered by the CBOR parser.

* suit-report-reason-cose-unsupported: an unsupported COSE ([RFC9052]) structure or header was encountered.

* suit-report-reason-alg-unsupported: an unsupported COSE algorithm was encountered.

* suit-report-reason-unauthorised: Signature/MAC verification failed.

* suit-report-reason-command-unsupported: an unsupported command was encountered.

* suit-report-reason-component-unsupported: The manifest declared a component/prefix that does not exist.

* suit-report-reason-component-unauthorised: The manifest declared a component that is not accessible by the signer.

* suit-report-reason-parameter-unsupported: The manifest used a parameter that does not exist.

* suit-report-reason-severing-unsupported: The manifest used severable fields but the Manifest Processor does not support them.

* suit-report-reason-condition-failed: A condition failed with soft-failure off.

* suit-report-reason-operation-failed: A command failed (e.g., download/copy/swap/write)

* suit-report-reason-invoke-pending: Invocation is about to be attempted and the final outcome is not yet known.

The suit-report-result-code reports an internal error code that is provided for debugging reasons. This code is not intended for interoperability.

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

NOTE: Some deployments use SUIT_Command_Invoke, which can transfer control to invoked code that never returns to the Manifest Processor. When a SUIT_Report is produced for remote attestation, implementations often need to sign the report before attempting the invoke. Signing with an unconditional "success" result would be misleading if the invocation ultimately fails. Implementers can leave the invoke outcome implicit—allowing a verifier to infer that execution was handed off—or, when the result must be reported before invocation, use suit-report-reason-invoke-pending to signal that invocation is about to occur without asserting a final outcome.

suit-report-capability-report provides a mechanism to report the capabilities of the Manifest Processor. The SUIT_Capability_Report is described in Section 6. The capability report is optional to include in the SUIT_Report, according to an application-specific policy. While the SUIT_Capability_Report is not expected to be very large, applications should ensure that they only report capabilities when necessary in order to conserve bandwidth. A capability report is not necessary except when:

1. A client explicitly requests the capability report, or

2. A manifest attempts to use a capability that the Manifest Processor does not implement.

5. Attestation

Where Remote Attestation (see [RFC9334], the RATS Architecture) is in use, the RATS Verifier (Verifier hereafter) requires a set of Attestation Evidence. Attestation Evidence contains Evidence Claims about the Attester. These Evidence Claims contain measurements about the Attester. Many of these measurements are the same measurements that are generated in SUIT, which means that a SUIT_Report contains most of the Claims and some of the Endorsements that a Verifier requires.

Using a SUIT_Manifest and a SUIT_Report improves a Verifier's ability to appraise the trustworthiness of a remote device. Remote attestation is done by using the SUIT_Envelope along with the SUIT_Report in Evidence to reconstruct the state of the device at boot time. Additionally, by including SUIT_Report data as telemetry (i.e., debug/failure information) next to measurements in Evidence, both types of Evidence data can be notarized via verifiable data structure, such as an append-only log (Section 3 of [I-D.ietf-scitt-architecture]) using the same conceptual message.

For the SUIT_Report to be usable as Attestation Evidence, the environment that generated the SUIT_Report also needs to be measured. Typically, this means that the software that executes the commands in the Manifest (the Manifest Processor) must be measured; similarly, the piece of software that assembles the measurements, taken by the Manifest Processor, into the SUIT_Report (the Report Generator) must also be measured. Any bootloaders or operating systems that facilitate the running of the Manifest Processor or Report Generator also need to be measured in order to demonstrate the integrity of the measuring environment.

Therefore, if a Remote Attestation format that conveys Attestation Evidence, such as an Entity Attestation Token (EAT, see [RFC9711]), contains a SUIT_Report, then it MUST also include an integrity measurement of the Manifest Processor, the Report Generator and any bootloader or OS environment that ran before or during the execution of both.

If Reference Values (Section 8.3 of [RFC9334]) required by the Verifier are delivered in a SUIT_Envelope, this codifies the delivery of appraisal information to the Verifier:

* The Firmware Distributor:

  - sends the SUIT_Envelope to the Verifier without payload or text, but with Reference Values

  - sends the SUIT_Envelope to the Recipient without Reference Values, 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 Reference Values

  - 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 signed Reference Values that is provided by the firmware author, which simplifies the delivery chain of verification information to the Verifier.

For a Verifier to consume the SUIT_Report, it requires a copy of the SUIT_Manifest. The Verifier then replays the SUIT_Manifest, using the SUIT_Report to resolve whether each condition is met. It identifies each measurement that is required by attestation policy and records this measurement as a Claim (Section 4 of [RFC9711]). It evaluates whether the SUIT_Report correctly matches the SUIT_Manifest as an element of evaluating trustworthiness. For example there are several indicators that would show that a SUIT_Report does not match a SUIT_Manifest. If any of the following (not an exhaustive list) occur, then the Manifest Processor that created the report is not trustworthy:

* Hash of SUIT_Manifest at suit-report-manifest-uri does not match suit-report-manifest-digest

* A SUIT_Record is issued for a SUIT_Command_Sequence that does not exist in the SUIT_Manifest at suit-report-manifest-uri.

* A SUIT_Record is identified at an offset that is not a condition and does not have a reporting policy that would indicate a SUIT_Record is needed.

Many architectures require multiple Verifiers, for example where one Verifier handles hardware trust, and another handles software trust, especially the evaluation of software authenticity and freshness. Some Verifiers may not be capable of processing a SUIT_Report and, for separation of roles, it may be preferable to divide that

responsibility. In this case, the Verifier of the SUIT_Report should perform an Evidence Transformation [I-D.ietf-rats-evidence-trans] and produce general purpose Measurement Results Claims that can be consumed by a downstream Verifier, for example a Verifying Relying Party, that does not understand SUIT_Reports.

6. Capability Reporting

Because SUIT is extensible, a manifest author must know what capabilities a device has available. To enable this, a capability report is a set of lists that define which commands, parameters, algorithms, and component IDs are supported by a manifest processor.

The CDDL for a SUIT_Capability_Report follows:

```
SUIT_Capability_Report = {
  suit-component-capabilities  => [+ SUIT_Component_Capability ]
  suit-command-capabilities          => [+ int],
  suit-parameters-capabilities       => [+ int],
  suit-crypt-algo-capabilities       => [+ int],
  ? suit-envelope-capabilities       => [+ int],
  ? suit-manifest-capabilities       => [+ int],
  ? suit-common-capabilities         => [+ int],
  ? suit-text-capabilities           => [+ int],
  ? suit-text-component-capabilities => [+ int],
  ? suit-dependency-capabilities     => [+ int],
  * [+int]                           => [+ int],
  $$SUIT_Capability_Report_Extensions
}

SUIT_Component_Capability = [*bstr,?true]
```

A SUIT_Component_Capability is similar to a SUIT_Component_ID, with one difference: it may optionally be terminated by a CBOR 'true' which acts as a wild-card match for any component with a prefix matching the SUIT_Component_Capability leading up to the 'true.' This feature is for use with filesystem storage, key value stores, or any other arbitrary-component-id storage systems.

When reporting capabilities, it is OPTIONAL to report capabilities that are declared mandatory by the SUIT Manifest [I-D.ietf-suit-manifest]. Capabilities defined by extensions MUST be reported.

Additional capability reporting can be added as follows: if a manifest element does not exist in this map, it can be added by specifying the CBOR path to the manifest element in an array and using this as the key. For example SUIT_Dependencies, as described

in Section 5.2.2 of [I-D.ietf-suit-trust-domains], could have an extension added, which was key 3 in the SUIT_Dependencies map. This capability would be reported as: [3, 3, 1] => [3], where the key consists of the key for SUIT_Manifest (3), the key for SUIT_Common (3), and the key for SUIT_Dependencies (1). Then the value indicates that this manifest processor supports the extension (3).

7. EAT Claim

The Entity Attestation Token (EAT, see [RFC9711]) is a secure container for conveying Attestation Evidence, such as measurements, and Attestation Results. The SUIT_Report is a form of measurement done by the SUIT Manifest Processor as it attempts to invoke a manifest or install a manifest. As a result, the SUIT_Report can be captured in an EAT measurements type.

The log-based structure of the SUIT_Report is not conducive to processing by a typical Relying Party: it contains only a list of waypoints through the SUIT Manifest--unless system parameter records are included--and requires additional information (the SUIT_Manifest) to reconstruct the values that must have been present at each test. A Verifier in posession of the SUIT_Manifest can reconstruct the measurements that would produce the waypoints in the SUIT_Report. The Verifier SHOULD convert a SUIT_Report into a more consumable version of the EAT claim by, for example, constructing a measurement results claim that contains the digest of a component, the Vendor ID and Class ID of a component, etc.

8. SUIT_Report Container

Transmission of the SUIT_Report MUST satisfy the requirements of Section 4.3.16 of [RFC9124]: REQ.SEC.REPORTING.

Status reports from the device to any remote system MUST be performed over an authenticated, confidential channel in order to prevent modification or spoofing of the reports.

As a result, the SUIT_Report MUST be transported using one of the following methods:

* As part of a larger document that provides authenticity guarantees, such as within a measurements claim in an Entity Attestation Token (EAT, Section 4.2.16 of [RFC9711]).

* As the payload of a message transmitted over a communication security protocol, such as DTLS [RFC9147].

* Encapsulated within a secure container, such as a COSE structure. In the case of COSE, the container MUST be either a COSE_Encrypt0 or COSE_Sign1 structure. The SUIT_Report MUST be the sole payload, as illustrated by the CDDL fragment below.

```
SUIT_Report_Protected /= SUIT_Report_COSE_Sign1 \
                         .and SUIT_COSE_Profiles
SUIT_Report_Protected /= SUIT_Report_COSE_Sign1_Tagged \
                         .and SUIT_COSE_Profiles
SUIT_Report_Protected /= SUIT_Report_COSE_MAC0 \
                         .and SUIT_COSE_Profiles
SUIT_Report_Protected /= SUIT_Report_COSE_MAC0_Tagged \
                         .and SUIT_COSE_Profiles

SUIT_Report_COSE_Sign1_Tagged = #6.18(SUIT_Report_COSE_Sign1)
SUIT_Report_COSE_Sign1 = [
    protected : bstr,
    unprotected : {* int => any},
    payload : bstr .cbor SUIT_Report_Unprotected,
    signature : bstr
]
SUIT_Report_COSE_MAC0_Tagged = #6.17(SUIT_Report_COSE_MAC0)
SUIT_Report_COSE_MAC0 = [
    protected : bstr,
    unprotected : {* int => any},
    payload : bstr .cbor SUIT_Report_Unprotected,
    tag : bstr
]
SUIT_Report_Unprotected = SUIT_Report / SUIT_Report_COSE_Encrypt0
SUIT_Report_COSE_Encrypt0 = COSE_Encrypt0
```

Note that SUIT_Report_COSE_Sign1 and SUIT_Report_COSE_MAC0 MUST be combined with a SUIT_COSE_Profiles from [I-D.ietf-suit-mti] using the CDDL .and directive. The SUIT_Report_COSE_Encrypt0 carries a ciphertext payload that MUST contain just the ciphertext obtained by encrypting the following CDDL:

```
SUIT_Report_plaintext = bstr .cbor SUIT_Report
```

SUIT_COSE_Profiles, which use AES-CTR encryption, are not integrity protected and authenticated. For this purpose, SUIT_Report_Protected defines authenticated containers with an encrypted payload.

9. IANA Considerations

IANA is requested to rename the overall SUIT registry group (https://www.iana.org/assignments/suit/suit.xhtml) "Software Update for the Internet of Things (SUIT)".

IANA is requested to allocate a CBOR tag for each of the following items. Please see Section 9.4 for further details.

* SUIT_Report_Protected

* SUIT_Reference

* SUIT_Capability_Report

IANA is requested to allocate a CoAP content-format [RFC7252] and a media-type for SUIT_Report Section 9.2. Please see Section 9.2 and Section 9.3 for further details.

IANA is also requested to add the following registries to the SUIT registry group (https://www.iana.org/assignments/suit/suit.xhtml).

* SUIT_Report Elements Section 9.5

* SUIT_Record Elements Section 9.6

* SUIT_Report Reasons Section 9.7

* SUIT_Capability_Report Elements Section 9.8

For each of these registries, registration policy is:

* -256 to 255: Standards Action

* -65536 to 257, 256 to 65535: Specification Required

* -4294967296 to -65537, 65536 to 4294967295: First Come First Served

Requests in the Standards Action and Specification Required ranges MUST also undergo designated expert review as described below; this guidance supplements the normal IANA processing for those policies.

9.1. Expert Review Instructions

The IANA registries established in this document allow values to be added based on expert review. This section gives some general guidelines for what the experts should be looking for, but they are being designated as experts for a reason, so they should be given substantial latitude.

Expert reviewers should take into consideration the following points:

* Point squatting should be discouraged. Reviewers are encouraged to get sufficient information for registration requests to ensure that the usage is not going to duplicate one that is already registered, and that the point is likely to be used in deployments. The zones tagged as private use are intended for testing purposes and closed environments; code points in other ranges should not be assigned for testing.

* Specifications are required for the standards track range of point assignment. Specifications should exist for all other ranges, but early assignment before a specification is available is considered to be permissible. When specifications are not provided, the description provided needs to have sufficient information to identify what the point is being used for.

* Experts should take into account the expected usage of fields when approving point assignment. The fact that there is a range for standards track documents does not mean that a standards track document cannot have points assigned outside of that range. The length of the encoded value should be weighed against how many code points of that length are left, the size of device it will be used on, and the number of code points left that encode to that size.

9.2. Media Type Registration

9.2.1. application/suit-report+cose

IANA is requested to register application/suit-report+cose as a media type for the SUIT_Report.

Type name: application
Subtype name: suit-report+cose
Required parameters: n/a
Encoding considerations: binary (CBOR)
Security considerations: Section 10 of RFCthis
Interoperability considerations: n/a
Published specification: RFCthis
Applications that use this media type: SUIT Manifest Processor, SUIT Manifest Distributor, SUIT Manifest Author, RATS Attesters, RATS Verifiers
Fragment identifier considerations: The syntax and semantics of fragment identifiers are as specified for "application/cose".
Person & email address to contact for further information: SUIT WG mailing list (suit@ietf.org)
Intended usage: COMMON
Restrictions on usage: none
Author/Change controller: IETF

Provisional registration: no

9.3. CoAP Content-Format Registration

IANA is requested to assign a CoAP Content-Format ID for the SUIT_Report media type in the "CoAP Content-Formats" registry, from the "IETF Review with Expert Review or IESG Approval with Expert Review" space (256..9999), within the "CoRE Parameters" registry group [RFC7252] [IANA.core-parameters]:

Content Type	Content Coding	ID	Reference
application/suit-report+cose		TBA	RFCthis

Table 1

9.4. CBOR Tag Registration

IANA is requested to allocate a tag in the "CBOR Tags" registry [IANA.cbor-tags], preferably in the Specification Required range:

Tag	Data Item	Semantics
TBA	array	SUIT_Report_Protected
TBA	array	SUIT_Reference
TBA	map	SUIT_Capability_Report

Table 2

9.5. SUIT_Report Elements

IANA is requested to create a new registry for SUIT_Report Elements.

Label	Name	CDDL Label	Reference
2	Nonce	suit-report-nonce	Section 4 of RFCthis
3	Records	suit-report-records	Section 4 of RFCthis
4	Result	suit-report-result	Section 4 of RFCthis
5	Result Code	suit-report-result-code	Section 4 of RFCthis
6	Result Record	suit-report-result-record	Section 4 of RFCthis
7	Result Reason	suit-report-result-reason	Section 4 of RFCthis
8	Capability Report	suit-report-capability-report	Section 4 of RFCthis
99	Reference	suit-reference	Section 4 of RFCthis

Table 3

9.6. SUIT_Record Elements

IANA is requested to create a new registry for SUIT_Record Elements.

Label	Name	CDDL Label	Reference
0	Manifest ID	suit-record-manifest-id	Section 3 of RFCthis
1	Manifest Section	suit-record-manifest-section	Section 3 of RFCthis
2	Section Offset	suit-record-section-offset	Section 3 of RFCthis
3	Component Index	suit-record-component-index	Section 3 of RFCthis
4	Record Properties	suit-record-properties	Section 3 of RFCthis

Table 4

9.7. SUIT_Report Reasons

IANA is requested to create a new registry for SUIT_Report Reasons.

Label	Name	CDDL Label	Reference
0	Result OK	suit-report-reason-ok	Section 4.2 of RFCthis
1	CBOR Parse Failure	suit-report-reason-cbor-parse	Section 4.2 of RFCthis
2	Unsupported COSE Structure or Header	suit-report-reason-cose-unsupported	Section 4.2 of RFCthis
3	Unsupported COSE Algorithm	suit-report-reason-alg-unsupported	Section 4.2 of RFCthis
4	Signature / MAC verification failed	suit-report-reason-unauthorised	Section 4.2 of RFCthis
5	Unsupported SUIT Command	suit-report-reason-command-unsupported	Section 4.2 of RFCthis
6	Unsupported SUIT Component	suit-report-reason-component-	Section 4.2 of RFCthis

		unsupported	
7	Unauthorized SUIT Component	suit-report-reason-component-unauthorised	Section 4.2 of RFCthis
8	Unsupported SUIT Parameter	suit-report-reason-parameter-unsupported	Section 4.2 of RFCthis
9	Severing Unsupported	suit-report-reason-severing-unsupported	Section 4.2 of RFCthis
10	Condition Failed	suit-report-reason-condition-failed	Section 4.2 of RFCthis
11	Operation Failed	suit-report-reason-operation-failed	Section 4.2 of RFCthis
12	Invocation Pending	suit-report-reason-invoke-pending	Section 4.2 of RFCthis

Table 5

9.8. SUIT Capability Report Elements

IANA is requested to create a new registry for SUIT Capability Report Elements.

Label	Name	Reference	
1	Components	suit-component-capabilities	Section 6 of RFCthis
2	Commands	suit-command-capabilities	Section 6 of RFCthis
3	Parameters	suit-parameters-capabilities	Section 6 of RFCthis
4	Cryptographic Algorithms	suit-crypt-algo-capabilities	Section 6 of RFCthis
5	Envelope Elements	suit-envelope-capabilities	Section 6 of RFCthis
6	Manifest Elements	suit-manifest-capabilities	Section 6 of RFCthis
7	Common Elements	suit-common-capabilities	Section 6 of RFCthis
8	Text Elements	suit-text-capabilities	Section 6 of RFCthis
9	Component Text Elements	suit-text-component-capabilities	Section 6 of RFCthis
10	Dependency Capabilities	suit-dependency-capabilities	Section 6 of RFCthis

Table 6

10. Security Considerations

The SUIT_Report serves four primary security objectives:

* Validated Identity
* Integrity
* Replay protection
* Confidentiality

The mechanisms for achieving these protections are outlined in Section 8.

Ideally, a SUIT_Report SHOULD be conveyed as part of a remote attestation procedure, such as embeding it in EAT tokens that represent RATS conceptual messages. This approach ensures that the SUIT_Report is cryptographically bound to the environment (hardware, software, or both) in which it was generated, thereby strengthening its authenticity.

A SUIT_Report may disclose sensitive information about the device on which it were produced. In such cases, the SUIT_Report MUST be encrypted, as specified in Section 8.

Furthermore, failure reports, particularly those involving cryptographic operations, can unintentionally reveal insights into system weaknesses or vulnerabilities. As such, SUIT_Reports SHOULD be encrypted whenever possible, to minimize the risk of information leakage.

In addition to these core security requirements, operational considerations must be taken into account. When a SUIT_Report is included within another protocol message (e.g., inside an encrypted EAT), care must be taken to avoid inadvertently leaking information and to uphold the principle of least privilege. For example, in many EAT-based remote attestation flows, the Verifier may not require the full SUIT_Report. Similarly, the Relying Party might not need access to it either.

To support least-privilege access, the SUIT_Report should be independently encrypted, even when the transport or enclosing token is also encrypted. This layered encryption ensures that only authorized entities can access the contents of the SUIT_Report.

In other scenarios, the EAT Verifier might require full access to a SUIT_Report. For example, the SUIT_Report must be accessible in its entirety for the EAT Verifier to extract or convert the SUIT_Report content into specific EAT claims, such as measres (Measurement Results). A typical case involves translating a successful suit-condition-image check into a digest-based claim within the EAT.

When applying cryptographic protection to the SUIT_Report, the same algorithm profile used for the corresponding SUIT manifest SHOULD be reused. The available algorithm profiles are detailed in [I-D.ietf-suit-mti]. If using the same profile is not feasible (e.g., due to constraints imposed by suit-sha256-hsslms-a256kw-a256ctr), then a profile offering comparable security strength SHOULD be selected—for instance, suit-sha256-esp256-ecdh-a128ctr.

In exceptional cases, if no suitable profile can be applied, the necessity of disabling a SUIT_Report functionality altogether might arise.

SUIT_Reports may expose information about the user to the Verifier, Firmware Distributor, or Manifest Author. Implementors MUST carefully consider user consent in the reporting system.

11. Acknowledgements

The authors would like to thank Dave Thaler for his feedback.

12. References

12.1. Normative References

[I-D.ietf-suit-manifest]
Moran, B., Tschofenig, H., Birkholz, H., Zandberg, K., and O. Rnningstad, "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-34, 28 May 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-suit-manifest-34>.

[I-D.ietf-suit-mti]
Moran, B., Rnningstad, O., and A. Tsukamoto, "Cryptographic Algorithms for Internet of Things (IoT) Devices", Work in Progress, Internet-Draft, draft-ietf-suit-mti-23, 22 July 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-suit-mti-23>.

[IANA.cbor-tags]
IANA, "Concise Binary Object Representation (CBOR) Tags", <https://www.iana.org/assignments/cbor-tags>.

[IANA.core-parameters]
IANA, "Constrained RESTful Environments (CoRE) Parameters", <https://www.iana.org/assignments/core-parameters>.

[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/rfc/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/rfc/rfc8174>.

[RFC8610] Birkholz, H., Vigano, C., and C. Bormann, "Concise Data Definition Language (CDDL): A Notational Convention to Express Concise Binary Object Representation (CBOR) and JSON Data Structures", RFC 8610, DOI 10.17487/RFC8610, June 2019, <https://www.rfc-editor.org/rfc/rfc8610>.

[RFC8792] Watsen, K., Auerswald, E., Farrel, A., and Q. Wu, "Handling Long Lines in Content of Internet-Drafts and RFCs", RFC 8792, DOI 10.17487/RFC8792, June 2020, <https://www.rfc-editor.org/rfc/rfc8792>.

[RFC9019] Moran, B., Tschofenig, H., Brown, D., and M. Meriac, "A Firmware Update Architecture for Internet of Things", RFC 9019, DOI 10.17487/RFC9019, April 2021, <https://www.rfc-editor.org/rfc/rfc9019>.

[RFC9052] Schaad, J., "CBOR Object Signing and Encryption (COSE): Structures and Process", STD 96, RFC 9052, DOI 10.17487/RFC9052, August 2022, <https://www.rfc-editor.org/rfc/rfc9052>.

[RFC9334] Birkholz, H., Thaler, D., Richardson, M., Smith, N., and W. Pan, "Remote ATtestation procedureS (RATS) Architecture", RFC 9334, DOI 10.17487/RFC9334, January 2023, <https://www.rfc-editor.org/rfc/rfc9334>.

[RFC9711] Lundblade, L., Mandyam, G., O’Donoghue, J., and C. Wallace, "The Entity Attestation Token (EAT)", RFC 9711, DOI 10.17487/RFC9711, April 2025, <https://www.rfc-editor.org/rfc/rfc9711>.

12.2. Informative References

[I-D.ietf-rats-evidence-trans]
Damato, F., Draper, A., and N. Smith, "Evidence Transformations", Work in Progress, Internet-Draft, draft-ietf-rats-evidence-trans-02, 17 October 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-rats-evidence-trans-02>.

[I-D.ietf-scitt-architecture]
Birkholz, H., Delignat-Lavaud, A., Fournet, C., Deshpande, Y., and S. Lasker, "An Architecture for Trustworthy and Transparent Digital Supply Chains", Work in Progress, Internet-Draft, draft-ietf-scitt-architecture-22, 10 October 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-scitt-architecture-22>.

[I-D.ietf-suit-trust-domains]
Moran, B. and K. Takayama, "Software Update for the Internet of Things (SUIT) Manifest Extensions for Multiple Trust Domain", Work in Progress, Internet-Draft, draft-ietf-suit-trust-domains-12, 22 July 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-suit-trust-domains-12>.

[RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, DOI 10.17487/RFC7252, June 2014, <https://www.rfc-editor.org/rfc/rfc7252>.

[RFC9124] Moran, B., Tschofenig, H., and H. Birkholz, "A Manifest Information Model for Firmware Updates in Internet of Things (IoT) Devices", RFC 9124, DOI 10.17487/RFC9124, January 2022, <https://www.rfc-editor.org/rfc/rfc9124>.

[RFC9147] Rescorla, E., Tschofenig, H., and N. Modadugu, "The Datagram Transport Layer Security (DTLS) Protocol Version 1.3", RFC 9147, DOI 10.17487/RFC9147, April 2022, <https://www.rfc-editor.org/rfc/rfc9147>.

Appendix A. Full CDDL

In order to create a valid SUIT_Report document the structure of the corresponding CBOR message MUST adhere to the following CDDL ([RFC8610]) data definition.

To be valid, the following CDDL MUST have the COSE CDDL appended to it. The COSE CDDL can be obtained by following the directions in Section 1.4 of [RFC9052]. It must also have the CDDL from [I-D.ietf-suit-mti] appended to it. This CDDL is line-wrapped per [RFC8792].

```
; NOTE: '\' line wrapping per RFC 8792
=============== NOTE: '\' line wrapping per RFC 8792 ================

SUIT_Report_Tool_Tweak /= SUIT_start
SUIT_Report_Tool_Tweak /= SUIT_Report_Protected
SUIT_Report_Protected /= SUIT_COSE_tool_tweak

SUIT_Report_Protected /= SUIT_Report_COSE_Sign1 .and \
                                                    SUIT_COSE_Profiles
SUIT_Report_Protected /= SUIT_Report_COSE_Sign1_Tagged .and \
                                                    SUIT_COSE_Profiles
SUIT_Report_Protected /= SUIT_Report_COSE_MAC0 .and \
                                                    SUIT_COSE_Profiles
SUIT_Report_Protected /= SUIT_Report_COSE_MAC0_Tagged .and \
                                                    SUIT_COSE_Profiles

SUIT_Report_COSE_Sign1_Tagged = #6.18(SUIT_Report_COSE_Sign1)
SUIT_Report_COSE_Sign1 = [
    protected : bstr,
    unprotected : {* int => any},
    payload : bstr .cbor SUIT_Report_Unprotected,
    signature : bstr
]
SUIT_Report_COSE_MAC0_Tagged = #6.17(SUIT_Report_COSE_MAC0)
SUIT_Report_COSE_MAC0 = [
    protected : bstr,
    unprotected : {* int => any},
    payload : bstr .cbor SUIT_Report_Unprotected,
    tag : bstr
]
SUIT_Report_Unprotected = SUIT_Report / SUIT_Report_COSE_Encrypt0
SUIT_Report_COSE_Encrypt0 = COSE_Encrypt0

SUIT_Report = {
  suit-reference              => SUIT_Reference,
  ? suit-report-nonce         => bstr,
  suit-report-records         => [
    * SUIT_Record / system-property-claims ],
  suit-report-result          => true / {
    suit-report-result-code   => int,
    suit-report-result-record => SUIT_Record,
    suit-report-result-reason => SUIT_Report_Reasons,
```

```
    },
    ? suit-report-capability-report => SUIT_Capability_Report,
    $$SUIT_Report_Extensions
  }

  SUIT_Reference = [
      suit-report-manifest-uri : tstr,
      suit-report-manifest-digest : SUIT_Digest
  ]

  SUIT_Record = [
      suit-record-manifest-id        : [* uint ],
      suit-record-manifest-section   : int,
      suit-record-section-offset     : uint,
      suit-record-component-index    : uint,
      suit-record-properties         : {*$$SUIT_Parameters},
      $$SUIT_Record_Extensions
  ]

  system-property-claims = {
    system-component-id => SUIT_Component_Identifier,
    + $$SUIT_Parameters,
  }

  SUIT_Capability_Report = {
    suit-component-capabilities  => [+ SUIT_Component_Capability]
    suit-command-capabilities          => [+ int],
    suit-parameters-capabilities       => [+ int],
    suit-crypt-algo-capabilities       => [+ int],
    ? suit-envelope-capabilities       => [+ int],
    ? suit-manifest-capabilities       => [+ int],
    ? suit-common-capabilities         => [+ int],
    ? suit-text-capabilities           => [+ int],
    ? suit-text-component-capabilities => [+ int],
    ? suit-dependency-capabilities     => [+ int],
    * [+int]                           => [+ int],
    $$SUIT_Capability_Report_Extensions
  }

  SUIT_Component_Capability = [*bstr,?true]

  suit-report-nonce = 2
  suit-report-records = 3
  suit-report-result = 4
  suit-report-result-code = 5
  suit-report-result-record = 6
  suit-report-result-reason = 7
  suit-report-capability-report = 8
```

```
   suit-reference = 99

   system-component-id = 0

   suit-record-manifest-id = 0
   suit-record-manifest-section = 1
   suit-record-section-offset = 2
   suit-record-component-index = 3
   suit-record-properties = 4

   SUIT_Report_Reasons /= suit-report-reason-ok
   SUIT_Report_Reasons /= suit-report-reason-cbor-parse
   SUIT_Report_Reasons /= suit-report-reason-cose-unsupported
   SUIT_Report_Reasons /= suit-report-reason-alg-unsupported
   SUIT_Report_Reasons /= suit-report-reason-unauthorised
   SUIT_Report_Reasons /= suit-report-reason-command-unsupported
   SUIT_Report_Reasons /= suit-report-reason-component-unsupported
   SUIT_Report_Reasons /= suit-report-reason-component-unauthorised
   SUIT_Report_Reasons /= suit-report-reason-parameter-unsupported
   SUIT_Report_Reasons /= suit-report-reason-severing-unsupported
   SUIT_Report_Reasons /= suit-report-reason-condition-failed
   SUIT_Report_Reasons /= suit-report-reason-operation-failed
   SUIT_Report_Reasons /= suit-report-reason-invoke-pending

   suit-report-reason-ok = 0
   suit-report-reason-cbor-parse = 1
   suit-report-reason-cose-unsupported = 2
   suit-report-reason-alg-unsupported = 3
   suit-report-reason-unauthorised = 4
   suit-report-reason-command-unsupported = 5
   suit-report-reason-component-unsupported = 6
   suit-report-reason-component-unauthorised = 7
   suit-report-reason-parameter-unsupported = 8
   suit-report-reason-severing-unsupported = 9
   suit-report-reason-condition-failed = 10
   suit-report-reason-operation-failed = 11
   suit-report-reason-invoke-pending = 12

   suit-component-capabilities        = 1
   suit-command-capabilities          = 2
   suit-parameters-capabilities       = 3
   suit-crypt-algo-capabilities       = 4
   suit-envelope-capabilities         = 5
   suit-manifest-capabilities         = 6
   suit-common-capabilities           = 7
   suit-text-capabilities             = 8
   suit-text-component-capabilities   = 9
   suit-dependency-capabilities       = 10
```

Authors' Addresses

Brendan Moran
Arm Limited
Email: brendan.moran.ietf@gmail.com

Henk Birkholz
Fraunhofer SIT
Rheinstrasse 75
64295 Darmstadt
Germany
Email: henk.birkholz@ietf.contact