Definition of the ROLIE Software Descriptor Extension

draft-banghart-sacm-rolie-softwaredescriptor-01

Abstract

This document extends the Resource-Oriented Lightweight Information Exchange (ROLIE) core to add the information type category and related requirements needed to support Software Record and Software Inventory use cases. The 'software-descriptor' information type is defined as a ROLIE extension. Additional supporting requirements are also defined that describe the use of specific formats and link relations pertaining to the new information type.

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 http://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 November 4, 2017.

Copyright Notice

Copyright (c) 2017 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust’s Legal Provisions Relating to IETF Documents (http://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of
1. Introduction

This document defines an extension to the Resource-Oriented Lightweight Information Exchange (ROLIE) protocol to support the publication of software descriptor information. Software descriptor information is information that characterizes:

an installable software package, or

information about static software components that may be installed by a software package or patch.

Software descriptor information includes identifying, versioning, software creation and publication, and file artifact information. Software descriptor information provides data about what might be
installed, but doesn’t describe where or how a specific software installation is installed, configured, or executed.

Some possible use cases for Software descriptor information include:

Software providers can publish software descriptor information so that software researchers and users of software can understand the collection of software produced by a that software provider.

Organizations can aggregate and syndicate collections of software descriptor information provided by multiple software providers to support software-related analysis processes (e.g., vulnerability analysis) and value added information (e.g., software configuration checklist repositories) using identification and characterization information derived from software descriptor information.

End user organizations can consume sources of software descriptor information, and other related software vulnerability and configuration information to provide the data needed to automate software asset, patch, and configuration management practices.

Organizations can use software descriptors to support verification of other entities, thru mechanisms such as RIM or other integrity measurements.

This document supports these use cases by describing the content requirements for Collections of software descriptor information that are to be published to or retrieved from a ROLIE repository. This document also discusses requirements around the use of link relationships and describing the data model formats used in a ROLIE Entry describing a software descriptor information resource.

2. Terminology

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

Definitions for some of the common computer security-related terminology used in this document can be found in Section 2 of [RFC5070].

3. New information-types

This document defines the following information type:
3.1. The "software-descriptor" information type

The "software-descriptor" information type represents any information that describes a piece of software. This document uses the definition of software provided by [RFC4949]. Note that as per this definition, this information type pertains to static software, that is, code on the disc. The software-descriptor information type is intended to provide a category for information that does one or more of the following:

identifies and characterizes software This software identification and characterization information can be provided by a large variety of data, but always describes software in a pre-installed state.

provides software installer metadata This represents information about software used to install other software. This metadata identifies, and characterizes a software installation package or media.

describes stateless installation metadata Information that describes the software post-deployment, such as files that may be deployed during an installation. It is expected that this metadata is produced generally for a given installation, and may not exactly match the actual installed files on a given endpoint.

Provided below is a non-exhaustive list of information that may be considered to be of a software-descriptor information type.

- Naming information: IDs and names that aid in the identification of a piece of software
- Version and patching information: Version numbers, patch identifiers, or other information that
- Vendor and source information: Includes where the software was developed or distributed from, as well as where the software installation media may be located.
- Payload and file information: information that describes or enumerates the files and folders that make up the piece of software, and information about those files.
- Descriptive information and data: Any information that otherwise characterizes a piece of software, such as libraries, runtime environments, target OSes, intended purpose or audience, etc.
Note again that this list is not exhaustive, any information that is in the abstract realm of an incident should be classified under this information-type.

This information type does not include descriptions of running software, or state and configuration information that is associated with a software installation.

4. Usage of CSIRT Information Types in the Atom Publishing Protocol

This document does not specify any additional requirements for use of the Atom Publishing Protocol.

5. Usage of the software-descriptor Information Type in the atom:feed element

This document does not specify any additional requirements for use of the atom:feed element.

6. Usage of the software-descriptor Information Type in an atom:entry

This document specifies the following requirements for use of the software-descriptor information type with regards to Atom Entries.

6.1. Use of the atom:link element

This section defines the requirements around the use of atom:links in Entries. Each relationship should be named, described, and given a requirement level.
Table 1: Link Relations for Resource-Oriented Lightweight Indicator Exchange

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Conformance</th>
</tr>
</thead>
<tbody>
<tr>
<td>ancestor</td>
<td>Links to a software descriptor resource that defines an ancestor of the software being described by this Entry.</td>
<td>MAY</td>
</tr>
<tr>
<td>patches</td>
<td>Links to a software descriptor resource that defines the software being patched by this software</td>
<td>MAY</td>
</tr>
<tr>
<td>requires</td>
<td>Links to a software descriptor resource that defines a piece of software required for this software to function properly.</td>
<td>MAY</td>
</tr>
<tr>
<td>installs</td>
<td>Links to a software descriptor resource that defines the software being installed by this software.</td>
<td>MAY</td>
</tr>
<tr>
<td>installationrecord</td>
<td>Provides a link to a resource that describes an installation of this software.</td>
<td>MAY</td>
</tr>
</tbody>
</table>

6.2. Use of the rolie:format element

This document does not contain any additional requirements for the rolie:format element, the formats that follow are provided as examples of formats that describe the software descriptor information type.

6.2.1. The ISO SWID 2016 format

The ISO SWID Tag 2016 format is a software descriptor and software record data format. It provides several tags: primary, which provides descriptive and naming information about software, patch, which describes non-standalone software meant to patch existing software, and corpus, which describes the software installation media that installs a given piece of software.

For a more complete overview as well as normative requirements, refer to TODO(ref?):ISO/IEC 19770-2
6.2.2. The Concise SWID format

The Concise SWID format is an alternative representation of the ISO SWID Tag 2016 format using a CBOR encoding defined by a CDDL specification. It provides the same features and attributes as are specified in ISO 19770-2, plus:

- a straightforward method to sign and encrypt SWID Tags using COSE, and
- additional attributes that provide an improved structure to include file hashes intended to be used as Reference Integrity Measurements (RIM).

6.3. Use of the rolie:property element

This document registers new valid rolie:property names as follows:

6.3.1. urn:ietf:params:rolie:property:swd:id

This property provides an exposure point for an identification field from the associated software descriptor. The value of this property SHOULD be uniquely identifying information generated from the software descriptor linked to by the entry’s atom:content element. swd:id property values SHOULD have a one-to-one mapping to individual pieces of SWD content.

6.3.2. urn:ietf:params:rolie:property:swd:swname

This property provides an exposure point for the plain text name of the software being described. Due to the great variance in naming schemes, this property should be considered informative.

6.4. IANA Considerations

6.4.1. incident information-type

IANA has added an entry to the "ROLIE Security Resource Information Type Sub-Registry" registry located at <https://www.iana.org/assignments/rolie/category/information-type>. The entry is as follows:

- name: software-descriptor
- index: TBD
- reference: This document, Section 3.1
6.4.2. swd:id property

IANA has added an entry to the "ROLIE URN Parameters" registry located in <https://www.iana.org/assignments/rolie/>.

The entry is as follows:

name: property:swd:id


Reference: This document, Section 6.3.1

Subregistry: None

6.4.3. swd:swname property

IANA has added an entry to the "ROLIE URN Parameters" registry located in <https://www.iana.org/assignments/rolie/>.

The entry is as follows:

name: property:swd:swname


Reference: This document, Section 6.3.2

Subregistry: None

6.5. Security Considerations

Use of this extension implies dealing with the security implications of both ROLIE and of software descriptors in general. As with any SWD information, care should be taken to verify the trustworthiness and veracity of the descriptor information to the fullest extent possible.

Ideally, software descriptors should have been signed by the software manufacturer, or signed by whichever agent processed the source code. SWD documents from these sources are more likely to be accurate than those generated by scraping installed software.

These "authoritative" sources of SWD content should consider additional security for their ROLIE repository beyond the typical recommendations, as the central importance of the repository is likely to make it a target.
Version information is often represented differently across manufacturers and even across product releases. If using SWD version information for low fault tolerance comparisons and searches, care should be taken that the correct version scheme is being utilized.

7. Normative References


Appendix A. Schema

This document does not require any schema extensions.

Appendix B. Examples of Use

Use of this extension in a ROLIE repository will not typically change that repo’s operation. As such, the general examples provided by the ROLIE core document would serve as examples. Provided below is a sample SWD ROLIE entry:
<entry xmlns="http://www.w3.org/2005/Atom"
  xmlns:rolie="urn:ietf:params:xml:ns:rolie-1.0">
  <id>dd786dba-88e6-440b-9158-b8fae67ef67c</id>
  <title>Sample Software Descriptor</title>
  <published>2015-08-04T18:13:51.0Z</published>
  <updated>2015-08-05T18:13:51.0Z</updated>
  <summary>A descriptor for a piece of software published by this
  organization. </summary>
  <link rel="self" href="http://www.example.org/provider/SWD/123456"/>
  <category
    scheme="urn:ietf:params:rolie:category:information-type"
    term="software-descriptor"/>
  <rolie:format ns="urn:example:COSWID"/>
  <content type="application/xml"
    src="http://www.example.org/provider/SWD/123456/data"/>
</entry>

Authors’ Addresses

David Waltermire
National Institute of Standards and Technology
100 Bureau Drive
Gaithersburg, Maryland 20877
USA

Email: david.waltermire@nist.gov

Stephen Banghart
National Institute of Standards and Technology
100 Bureau Drive
Gaithersburg, Maryland 20877
USA

Email: stephen.banghart@nist.gov
Concise Software Identifiers
draft-ietf-sacm-coswid-06

Abstract

This document defines a concise representation of ISO/IEC 19770-2:2015 Software Identification (SWID) tags that are interoperable with the XML schema definition of ISO/IEC 19770-2:2015 and augmented for application in Constrained-Node Networks. Next to the inherent capability of SWID tags to express arbitrary context information, Concise SWID (CoSWID) tags support the definition of additional semantics via well-defined data definitions incorporated by extension points.

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 4, 2019.

Copyright Notice

Copyright (c) 2018 IETF Trust and the persons identified as the document authors. All rights reserved.
This document is subject to BCP 78 and the IETF Trust’s Legal Provisions Relating to IETF Documents (https://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction ............................................. 3
   1.1. The SWID Tag Lifecycle .............................. 4
   1.2. Concise SWID Extensions ............................. 6
   1.3. Requirements Notation ................................ 7
2. Concise SWID Data Definition ............................. 7
   2.1. The concise-software-identity Object .................. 8
       2.1.1. Determining the tag type ....................... 11
       2.1.2. concise-software-identity Co-constraints .......... 12
   2.2. The global-attributes Group ........................... 12
   2.3. The any-element-map Entry ............................ 13
   2.4. The entity Object ................................... 13
   2.5. The link Object ..................................... 14
   2.6. The software-meta Object ............................. 16
   2.7. The Resource Collection Definition .................. 19
       2.7.1. The hash-entry Array ............................ 19
       2.7.2. The resource-collection Group .................... 20
       2.7.3. The payload Object ............................... 22
       2.7.4. The evidence Object .............................. 23
   2.8. Full CDDL Definition .................................. 24
3. CoSWID Indexed Label Values ............................. 29
   3.1. Version Scheme ..................................... 29
   3.2. Entity Role Values .................................. 29
4. IANA Considerations ..................................... 30
   4.1. SWID/CoSWID Version Schema Values Registry ........... 30
   4.2. SWID/CoSWID Entity Role Values Registry .............. 31
5. Security Considerations .................................. 32
6. Acknowledgments ....................................... 34
7. Change Log .............................................. 34
8. Contributors ............................................ 36
9. References .............................................. 36
   9.1. Normative References ................................ 36
   9.2. Informative References .............................. 37
Appendix A. CoSWID Attributes for Firmware (label 60) .......... 38
Appendix B. Signed Concise SWID Tags using COSE .................. 44
Appendix C. CoSWID used as Reference Integrity Measurements (CoSWID RIM) .................. 45
1. Introduction

SWID tags have several use-applications including but not limited to:

- Software Inventory Management, a part of the Software Asset Management [SAM] process, which requires an accurate list of discernible deployed software components.

- Vulnerability Assessment, which requires a semantic link between standardized vulnerability descriptions and software components installed on IT-assets [X.1520].

- Remote Attestation, which requires a link between reference integrity measurements (RIM) and security logs of measured software components [I-D.birkholz-tuda].

SWID tags, as defined in ISO-19770-2:2015 [SWID], provide a standardized XML-based record format that identifies and describes a specific release of a software component. Different software components, and even different releases of a particular software component, each have a different SWID tag record associated with them. SWID tags are meant to be flexible and able to express a broad set of metadata about a software component.

While there are very few required fields in SWID tags, there are many optional fields that support different use scenarios. While a SWID tag consisting of only required fields might be a few hundred bytes in size, a tag containing many of the optional fields can be many orders of magnitude larger. Thus, real-world instances of SWID tags can be fairly large, and the communication of SWID tags in use-applications such as those described earlier can cause a large amount of data to be transported. This can be larger than acceptable for constrained devices and networks. Concise SWID (CoSWID) tags significantly reduce the amount of data transported as compared to a typical SWID tag. This reduction is enabled through the use of CBOR, which maps human-readable labels of that content to more concise integer labels (indices). The use of CBOR to express SWID information in CoSWID tags allows both CoSWID and SWID tags to be part of an enterprise security solution for a wider range of endpoints and environments.
1.1. The SWID Tag Lifecycle

In addition to defining the format of a SWID tag record, ISO/IEC 19770-2:2015 defines requirements concerning the SWID tag lifecycle. Specifically, when a software component is installed on an endpoint, that product’s SWID tag is also installed. Likewise, when the product is uninstalled or replaced, the SWID tag is deleted or replaced, as appropriate. As a result, ISO/IEC 19770-2:2015 describes a system wherein there is a correspondence between the set of installed software components on an endpoint, and the presence of the corresponding SWID tags for these components on that endpoint. CoSWIDs share the same lifecycle requirements as a SWID tag.

The following is an excerpt (with some modifications and reordering) from NIST Interagency Report (NISTIR) 8060: Guidelines for the Creation of Interoperable SWID Tags [SWID-GUIDANCE], which describes the tag types used within the lifecycle defined in ISO-19770-2:2015.

The SWID specification defines four types of SWID tags: primary, patch, corpus, and supplemental.

1. Primary Tag - A SWID or CoSWID tag that identifies and describes a software component is installed on a computing device.

2. Patch Tag - A SWID or CoSWID tag that identifies and describes an installed patch which has made incremental changes to a software component installed on a computing device.

3. Corpus Tag - A SWID or CoSWID tag that identifies and describes an installable software component in its pre-installation state. A corpus tag can be used to represent metadata about an installation package or installer for a software component, a software update, or a patch.

4. Supplemental Tag - A SWID or CoSWID tag that allows additional information to be associated with a referenced SWID tag. This helps to ensure that SWID Primary and Patch Tags provided by a software provider are not modified by software management tools, while allowing these tools to provide their own software metadata.

Corpus, primary, and patch tags have similar functions in that they describe the existence and/or presence of different types of software (e.g., software installers, software installations, software patches), and, potentially, different states of software components. In contrast, supplemental tags furnish additional information not contained in corpus, primary, or patch tags. All
four tag types come into play at various points in the software lifecycle, and support software management processes that depend on the ability to accurately determine where each software component is in its lifecycle.

```
+-------------+
| v            |
| Installation | Product     | Product | Product | Product |
| Media        | -> Installed | -> Patched | -> Upgraded | -> Removed |
| Deployed     |             |           |           |          |
| Corpus       | Primary     | Primary   | xPrimary | xPrimary |
| Supplemental | Supplemental| xSupplemental | xSupplemental |
| Patch        | xPatch      | Primary   |          |
|              | Supplemental|          |          |
```

**Figure 1: Use of Tag Types in the Software Lifecycle**

Figure 1 illustrates the steps in the software lifecycle and the relationships among those lifecycle events supported by the four types of SWID and CoSWID tags, as follows:

* **Software Deployment.** Before the software component is installed (i.e., pre-installation), and while the product is being deployed, a corpus tag provides information about the installation files and distribution media (e.g., CD/DVD, distribution package).

* **Software Installation.** A primary tag will be installed with the software component (or subsequently created) to uniquely identify and describe the software component. Supplemental tags are created to augment primary tags with additional site-specific or extended information. While not illustrated in the figure, patch tags may also be installed during software installation to provide information about software fixes deployed along with the base software installation.

* **Software Patching.** When a new patch is applied to the software component, a new patch tag is provided, supplying details about the patch and its dependencies. While not illustrated in the figure, a corpus tag can also provide information about the patch installer, and patching dependencies that need to be installed before the patch.
Software Upgrading. As a software component is upgraded to a new version, new primary and supplemental tags replace existing tags, enabling timely and accurate tracking of updates to software inventory. While not illustrated in the figure, a corpus tag can also provide information about the upgrade installer, and dependencies that need to be installed before the upgrade.

Software Removal. Upon removal of the software component, relevant SWID tags are removed. This removal event can trigger timely updates to software inventory reflecting the removal of the product and any associated patch or supplemental tags.

Note: While not fully illustrated in the figure, supplemental tags can be associated with any corpus, primary, or patch tag to provide additional metadata about an installer, installed software, or installed patch respectively.

Each of the different SWID and CoSWID tag types provide different sets of information. For example, a "corpus tag" is used to describe a software component’s installation image on an installation media, while a "patch tag" is meant to describe a patch that modifies some other software component.

1.2. Concise SWID Extensions

This document defines the CoSWID format, a more concise representation of SWID information in the Concise Binary Object Representation (CBOR) [RFC7049]. This is described via the Concise Data Definition Language (CDDL) [I-D.ietf-cbor-cddl]. The resulting CoSWID data definition is interoperable with the XML schema definition of ISO-19770-2:2015 [SWID]. The vocabulary, i.e., the CDDL names of the types and members used in the CoSWID data definition, are mapped to more concise labels represented as small integer values. The names used in the CDDL data definition and the mapping to the CBOR representation using integer labels is based on the vocabulary of the XML attribute and element names defined in ISO/IEC 19770-2:2015.

The corresponding CoSWID data definition includes two kinds of augmentation.

- The explicit definition of types for attributes that are typically stored in the "any attribute" of an ISO-19770-2:2015 in XML representation. These are covered in Section 2.2 and Section 2.3 of this document.
The inclusion of extension points in the CoSWID data definition that allow for additional uses of CoSWID tags that go beyond the original scope of ISO-19770-2:2015 tags. These are covered in Section 2.7.3 and Section 2.7.4.

1.3. Requirements Notation

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, BCP 14 [RFC2119].

2. Concise SWID Data Definition

The following is a CDDL representation for a CoSWID tag. This CDDL representation is intended to be parallel to the XML schema definition in the ISO/IEC 19770-2:2015 [SWID] specification, allowing both SWID and CoSWID tags to represent a common set of SWID information and to support all SWID tag use cases. To achieve this end, the CDDL representation includes every SWID tag field and attribute. The CamelCase notation used in the XML schema definition is changed to a hyphen-separated notation (e.g. ResourceCollection is named resource-collection in the CoSWID data definition). This deviation from the original notation used in the XML representation reduces ambiguity when referencing certain attributes in corresponding textual descriptions. An attribute referred by its name in CamelCase notation explicitly relates to XML SWID tags, an attribute referred by its name in hyphen-separated notation explicitly relates to CoSWID tags. This approach simplifies the composition of further work that reference both XML SWID and CoSWID documents.

Human-readable names of members in the CDDL data definition are mapped to integer indices via a block of rules at the bottom of the definition. The 67 character strings of the SWID vocabulary that would have to be stored or transported in full if using the original vocabulary are replaced.

In CBOR, an array is encoded using bytes that identify the array, and the array's length or stop point (see [RFC7049]). To make items that support 1 or more values, the following CDDL notion is used.

\_name\_ = (\_label\_: \_data\_ / [ 2* \_data\_ ])

The CDDL above allows for a more efficient CBOR encoding of the data when a single value is used by avoiding the need to first encode the array. An array is used for two or more values. This modeling pattern is used frequently in the CoSWID CDDL data definition in such cases.
The following subsections describe the different parts of the CoSWID model.

2.1. The concise-software-identity Object

The CDDL for the main concise-software-identity object is as follows:

concrete-software-identity = {
  global-attributes,
  tag-id,
  tag-version,
  ? corpus,
  ? patch,
  ? supplemental,
  swid-name,
  ? software-version,
  ? version-scheme,
  ? media,
  ? software-meta-entry,
  ? entity-entry,
  ? link-entry,
  ? ( payload-entry / evidence-entry ),
  ? any-element-entry,
}

tag-id = (0: text)
swid-name = (1: text)
entity-entry = (2: entity / [ 2* entity ])
evidence-entry = (3: evidence)
link-entry = (4: link / [ 2* link ])
software-meta-entry = (5: software-meta / [ 2* software-meta ])
payload-entry = (6: payload)
any-element-entry = (7: any-element-map / [ 2* any-element-map ])
corpus = (8: bool)
patch = (9: bool)
media = (10: text)
supplemental = (11: bool)
tag-version = (12: integer)
software-version = (13: text)
version-scheme = (14: text)

The following describes each child item of the concise-software-identity object model.

- global-attributes: A list of items including an optional language definition to support the processing of text-string values and an unbounded set of any-attribute items. Described in Section 2.2.
o tag-id (label 0): An textual identifier uniquely referencing a (composite) software component. The tag identifier MUST be globally unique. There are no strict guidelines on how this identifier is structured, but examples include a 16 byte GUID (e.g. class 4 UUID) [RFC4122].

o tag-version (label 12): An integer value that indicates if a specific release of a software component has more than one tag that can represent that specific release. Typically, the initial value of this field is set to 0, and the value is monotonically increased for subsequent tags produced for the same software component release. This item is used when a CoSWID tag producer creates and releases an incorrect tag that they subsequently want to fix, but no underlying changes have been made to the product the CoSWID tag represents. This could happen if, for example, a patch is distributed that has a link reference that does not cover all the various software releases it can patch. A newer CoSWID tag for that patch can be generated and the tag-version value incremented to indicate that the data is updated.

o corpus (label 8): A boolean value that indicates if the tag identifies and describes an installable software component in its pre-installation state. Installable software includes a installation package or installer for a software component, a software update, or a patch. If the CoSWID tag represents installable software, the corpus item MUST be set to "true". If not provided the default value MUST be considered "false".

o patch (label 9): A boolean value that indicates if the tag identifies and describes an installed patch which has made incremental changes to a software component installed on a computing device. Typically, an installed patch has made a set of file modifications to pre-installed software, and does not alter the version number or the descriptive metadata of an installed software component. If a CoSWID tag is for a patch, it MUST contain the patch item and its value MUST be set to "true". If not provided the default value MUST be considered "false".

o supplemental (label 11): A boolean value that indicates if the tag is providing additional information to be associated with another referenced SWID or CoSWID tag. Tags using this item help to ensure that primary and patch tags provided by a software provider are not modified by software management tools, while allowing these tools to provide their own software metadata for a software component. If a CoSWID tag is a supplemental tag, it MUST contain the supplemental item and its value MUST be set to "true". If not provided the default value MUST be considered "false".
- **swid-name (label 1)**: This textual item provides the software component name as it would typically be referenced. For example, what would be seen in the add/remove software dialog in an operating system, or what is specified as the name of a packaged software component or a patch identifier name.

- **software-version (label 13)**: A textual value representing the specific underlying release or development version of the software component.

- **version-scheme (label 14)**: An 8-bit integer or textual value representing the versioning scheme used for the software-version item. If an integer value is used it MUST be a value from the registry (see section Section 4.1 or a value in the private use range: 32768-65,535.

- **media (label 10)**: This text value is a hint to the tag consumer to understand what this tag applies to. This item represents a query as defined by the W3C Media Queries Recommendation (see http://www.w3.org/TR/css3-mediaqueries/). A hint to the consumer of the link to what the target item is applicable for.

- **software-meta-entry (label 5)**: An open-ended collection of key/value data related to this CoSWID. A number of predefined attributes can be used within this item providing for common usage and semantics across the industry. The data definition of this entry allows for any additional attribute to be included, though it is recommended that industry norms for new attributes are defined and followed to the degree possible. Described in Section 2.6.

- **entity-entry (label 2)**: Specifies the organizations related to the software component referenced by this CoSWID tag. Described in Section 2.4.

- **link-entry (label 4)**: Provides a means to establish a relationship arc between the tag and another item. A link can be used to establish relationships between tags and to reference other resources that are related to the CoSWID tag, e.g. vulnerability database associations, ROLIE feeds, MUD files, software download location, etc). This is modeled after the HTML "link" element. Described in Section 2.5.

- **payload-entry (label 6)**: The items that may be installed on a system entity when the software component is installed. Note that payload may be a superset of the items installed and - depending on optimization mechanisms in respect to that system entity - may or may not include every item that could be created or executed on
the corresponding system entity when software components are installed. In general, payload will be used to indicate the files that may be installed with a software component. Therefore payload will often be a superset of those files (i.e. if a particular optional sub-component is not installed, the files associated with that software component may be included in payload, but not installed in the system entity). Described in Section 2.7.3.

- evidence-entry (label 3): This item is used to provide results from a scan of a system where software that does not have a CoSWID tag is discovered. This information is not provided by the software-creator, and is instead created when a system is being scanned and the evidence for why software is believed to be installed on the device is provided in the evidence item. Described in Section 2.7.4.

- any-element-entry (label 7): A default map that can contain arbitrary map members and even nested maps (which would also be any-elements). In essence, the any-element allows items not defined in this CDDL data definition to be included in a Concise Software Identifier. Described in Section 2.3.

### 2.1.1. Determining the tag type

The operational model for SWID and CoSWID tags introduced in Section 1.1. The following rules can be used to determine the type of a CoSWID tag.

- Corpus Tag: A CoSWID tag MUST be considered a corpus tag if the corpus item is "true".

- Primary Tag: A CoSWID tag MUST be considered a primary tag if the corpus, patch, and supplemental items are "false".

- Patch Tag: A CoSWID tag MUST be considered a patch tag if the patch item is "true" and the corpus item is "false".

- Supplemental Tag: A CoSWID tag MUST be considered a supplemental tag if the supplemental item is set to "true".

A tag that does not match one of the above rules MUST be considered an invalid, unsupported tag type.

If a patch modifies the version number or the descriptive metadata of the software, then a new tag representing these details SHOULD be installed, and the old tag SHOULD be removed.
2.1.2. concise-software-identity Co-constraints

- Only one of the corpus, patch, and supplemental items MUST be set to "true", or all of the corpus, patch, and supplemental items MUST be set to "false" or be omitted.

- If the patch item is set to "true", the the tag SHOULD contain at least one link with the rel(ation) item value of "patches" and an href item specifying an association with the software that was patched.

- If the supplemental item is set to "true", the the tag SHOULD contain at least one link with the rel(ation) item value of "supplements" and an href item specifying an association with the software that is supplemented.

- If all of the corpus, patch, and supplemental items are "false", or if the corpus item is set to "true", then a software-version item MUST be included with a value set to the version of the software component. This ensures that primary and corpus tags have an identifiable software version.

2.2. The global-attributes Group

The global-attributes group provides a list of items including an optional language definition to support the processing of text-string values and an unbounded set of any-attribute items allowing for additional items to be provided as a general point of extension in the model.

The CDDL for the global-attributes is as follows:

```
global-attributes = (  
    ? lang,  
    * any-attribute, 
)  
label = text / int  
any-attribute = (  
    label => text / int / [ 2* text ] / [ 2* int ]  
)  
lang = (15: text)  
```

The following describes each child item of this object.
o  lang (index 15): A language tag or corresponding IANA index integer that conforms with IANA Language Subtag Registry [RFC5646].

o  any-attribute: This sub-group provides a means to include arbitrary information via label (key) item value pairs where both keys and values can be either a single integer or text string, or an array of integers or text strings.

2.3. The any-element-map Entry

The CDDL for the any-element-entry object is as follows:

any-element-map = {
    global-attributes,
    * label => any-element-map / [ 2* any-element-map ],
} 

any-element-entry = (7: any-element-map / [ 2* any-element-map ])

The following describes each child item of this object.

o  global-attributes: The global-attributes group described in Section 2.2.

o  label: a single or multiple

2.4. The entity Object

The CDDL for the entity object is as follows:

text

extended-data = (30: any-element-map / [ 2* any-element-map ])

entity-name = (31: text)

reg-id = (32: any-uri)

text

role = (33: text / [2* text])

thumbprint = (34: hash-entry)

The following describes each child item of this object.
global-attributes: The global-attributes group described in Section 2.2.

entity-name (index 32): The text-string name of the organization claiming a particular role in the CoSWID tag.

reg-id (index 32): The registration id is intended to uniquely identify a naming authority in a given scope (e.g. global, organization, vendor, customer, administrative domain, etc.) that is implied by the referenced naming authority. The value of an registration ID MUST be a RFC 3986 URI. The scope SHOULD be the scope of an organization. In a given scope, the registration id MUST be used consistently.

role (index 33): The relationship(s) between this organization and this tag. The role of tag creator is required for every CoSWID tag. The role of an entity may include any role value, but the pre-defined roles include: "aggregator", "distributor", "licensor", "software-creator", and "tag-creator". These pre-defined role index and text values are defined in Section 3.2. Use of index values instead of text for these pre-defined roles allows a CoSWID to be more concise.

thumbprint (index 34): The value of the thumbprint item provides an integer-based hash algorithm identifier (hash-alg-id) and a byte string string value (hash-value) that contains the corresponding hash value (i.e. the thumbprint) of the signing entities certificate(s). If the hash-alg-id is not known, then the integer value "0" MUST be used. This ensures parity between the SWID tag specification [SWID], which does not allow an algorithm to be identified for this field. See Section 2.7.1 for more details on the use of the hash-entry data structure.

extended-data (index 30): An open-ended collection of elements that can be used to attach arbitrary metadata to an entity item.

2.5. The link Object

The CDDL for the link object is as follows:
The following describes each child item of this object.

- **global-attributes**: The global-attributes group described in Section 2.2.

- **artifact (index: 37)**: For installation media (rel="installation-media"), this item value indicates the path of the installer executable or script that can be run to launch the referenced installation. Items with the same artifact name should be considered mirrors of each other, allowing the installation media to be downloaded from any of the described sources.

- **href (index 38)**: The link to the item being referenced. The "href" item’s value can point to several different things, and can be any of the following:
  
  * If no URI scheme is provided, then the URI is to be interpreted as being relative to the URI of the CoSWID tag. For example, "./folder/supplemental.coswid".
  
  * a physical resource location with any system-acceptable URI scheme (e.g., file:// http:// https:// ftp://)
  
  * a URI with "coswid:" as the scheme, which refers to another CoSWID by tag-id. This URI would need to be resolved in the context of the system by software that can lookup other CoSWID tags. For example, "coswid:2df9de35-0aff-4a86-ace6-f7dddlade4c" references the tag with the tag-id value "2df9de35-0aff-4a86-ace6-f7dddlade4c".
a URI with "swidpath:" as the scheme, which refers to another CoSWID via an XPATH query. This URI would need to be resolved in the context of the system entity via dedicated software components that can lookup other CoSWID tags and select the appropriate tag based on an XPATH query. Examples include:

```
swidpath://SoftwareIdentity[Entity/@regid='http://contoso.com']
```

would retrieve all CoSWID tags that include an entity where the regid is "Contoso" or

```
swidpath://SoftwareIdentity[Meta/@persistentId='b0c55172-38e9-4e36-be86-92206ad8edd8']
```

would match CoSWID tags with the persistent-id value "b0c55172-38e9-4e36-be86-92206ad8edd8".

* See XPATH query standard: http://www.w3.org/TR/xpath20/

- media (index 10): See media defined in Section 2.1.

- ownership (index 39): Determines the relative strength of ownership of the software components. Valid enumerations are: abandon, private, shared

- rel (index 40): The relationship between this CoSWID and the target file. Relationships can be identified by referencing the IANA registration library: https://www.iana.org/assignments/link-relations/link-relations.xhtml.

- media-type (index 41): The IANA MediaType for the target file; this provides the consumer with intelligence of what to expect. See http://www.iana.org/assignments/media-types/media-types.xhtml for more details on link type.

- use (index 42): Determines if the target software is a hard requirement or not. Valid enumerations are: required, recommended, optional.

2.6. The software-meta Object

The CDDL for the software-meta object is as follows:
software-meta = {
  global-attributes,
  ? activation-status,
  ? channel-type,
  ? colloquial-version,
  ? description,
  ? edition,
  ? entitlement-data-required,
  ? entitlement-key,
  ? generator,
  ? persistent-id,
  ? product,
  ? product-family,
  ? revision,
  ? summary,
  ? unspsc-code,
  ? unspsc-version,
}
activation-status = (43: text)
channel-type = (44: text)
colloquial-version = (45: text)
description = (46: text)
edition = (47: text)
extitlement-data-required = (48: bool)
extitlement-key = (49: text)
generator = (50: text)
persistent-id = (51: text)
product = (52: text)
product-family = (53: text)
revision = (54: text)
summary = (55: text)
unspsc-code = (56: text)
unspsc-version = (57: text)

The following describes each child item of this object.

  o global-attributes: The global-attributes group described in Section 2.2.

  o activation-status (index 43): Identification of the activation status of this software title (e.g. Trial, Serialized, Licensed, Unlicensed, etc). Typically, this is used in supplemental tags.

  o channel-type (index 44): Provides information on which channel this particular software was targeted for (e.g. Volume, Retail, OEM, Academic, etc). Typically used in supplemental tags.
o colloquial-version (index 45): The informal or colloquial version of the product (i.e. 2013). Note that this version may be the same through multiple releases of a software component where the version specified in entity is much more specific and will change for each software release. Note that this representation of version is typically used to identify a group of specific software releases that are part of the same release/support infrastructure (i.e. Fabrikam Office 2013). This version is used for string comparisons only and is not compared to be an earlier or later release (that is done via the entity version).

o description (index 46): A longer, detailed description of the software. This description can be multiple sentences (differentiated from summary, which is a very short, one-sentence description).

o edition (index 47): The variation of the product (Extended, Enterprise, Professional, Standard etc).

o entitlement-data-required (index 48): An indicator to determine if there should be accompanying proof of entitlement when a software license reconciliation is completed.

o entitlement-key (index 49): A vendor-specific textual key that can be used to reconcile the validity of an entitlement. (e.g. serial number, product or license key).

o generator (index 50): The name of the software tool that created a CoSWID tag. This item is typically used if tags are created on the fly or via a catalog-based analysis for data found on a computing device.

o persistent-id (index 51): A GUID used to represent products installed where the products are related, but may be different versions.

o product (index 52): The base name of the product (e.g.).

o product-family (index 53): The overall product family this software belongs to. Product family is not used to identify that a product is part of a suite, but is instead used when a set of products that are all related may be installed on multiple different devices. For example, an enterprise backup system may consist of a backup services, multiple different backup services that support mail services, databases and ERP systems, as well as individual software components that backup client system entities. In such an usage scenario, all software components that are part
of the backup system would have the same product-family name so they can be grouped together in respect to reporting systems.

- revision (index 54): The informal or colloquial representation of the sub-version of the given product (i.e., SP1, R2, RC1, Beta 2, etc). Note that the version will provide very exact version details, the revision is intended for use in environments where reporting on the informal or colloquial representation of the software is important (for example, if for a certain business process, an organization recognizes that it must have, for example "ServicePack 1" or later of a specific product installed on all devices, they can use the revision data value to quickly identify any devices that do not meet this requirement). Depending on how a software organization distributes revisions, this value could be specified in a primary (if distributed as an upgrade) or supplemental (if distributed as a patch) CoSWID tag.

- summary (index 55): A short (one-sentence) description of the software.

- unspsc-code (index 56): An 8 digit code that provides UNSPSC classification of the software component this SWID tag identifies. For more information see, http://www.unspsc.org/.

- unspsc-version (index 57): The version of the UNSPSC code used to define the UNSPSC code value. For more information see, http://www.unspsc.org/.

2.7. The Resource Collection Definition

2.7.1. The hash-entry Array

CoSWID add explicit support for the representation of hash entries using algorithms that are registered at the Named Information Hash Algorithm Registry via the hash-entry member (label 58).

hash-entry = (58: [ hash-alg-id: int, hash-value: bstr ] )

The number used as a value for hash-alg-id MUST refer an ID in the Named Information Hash Algorithm Registry; other hash algorithms MUST NOT be used. The hash-value MUST represent the raw hash value of the hashed resource generated using the hash algorithm indicated by the hash-alg-id.
2.7.2. The resource-collection Group

A list of items both used in evidence (discovered by an inventory process) and payload (installed in a system entity) content of a CoSWID tag document to structure and differentiate the content of specific CoSWID tag types. Potential content includes directories, files, processes, resources or firmwares.

The CDDL for the resource-collection group is as follows:

```cddl
resource-collection = {
  ? directory-entry,
  ? file-entry,
  ? process-entry,
  ? resource-entry
}

directory = {
  filesystem-item,
  path-elements,
}

file = {
  filesystem-item,
  ? size,
  ? file-version,
  ? hash-entry,
}

process = {
  global-attributes,
  process-name,
  ? pid,
}

resource = {
  global-attributes,
  type,
}

filesystem-item = {
  global-attributes,
  ? key,
  ? location,
  fs-name,
  ? root,
}```
directory-entry = (16: directory / [ 2* directory ])
file-entry = (17: file / [ 2* file ])
process-entry = (18: process / [ 2* process ])
resource-entry = (19: resource / [ 2* resource ])
size = (20: integer)
file-version = (21: text)
key = (22: bool)
location = (23: text)
fs-name = (24: text)
root = (25: text)
path-elements = (26: { * file-entry,
                    * directory-entry,
                    })
process-name = (27: text)
pid = (28: integer)
type = (29: text)

The following describes each child item or group for these groups.

- filesystem-item: A list of items both used in representing the
  nodes of a file-system hierarchy, i.e. directory items that allow
  one or more directories to be defined in the file structure, and
  file items that allow one or more files to be specified for a
  given location.

- global-attributes: The global-attributes group described in
  Section 2.2.

- directory-entry (index 16): A directory item allows one or more
  directories to be defined in the file structure.

- file-entry (index 17): A file element that allows one or more
  files to be specified for a given location.

- process-entry (index 18): Provides process (software component in
  execution) information for data that will show up in a devices
  process table.

- resource-entry (index 19): A set of items that can be used to
  provide arbitrary resource information about an application
  installed on a system entity, or evidence collected from a system
  entity.

- size (index 20): The file size in bytes of the file.

- file-version (index 21): The version of the file.
key (index 22): Files that are considered important or required for the use of a software component. Typical key files would be those which, if not available on a system entity, would cause the software component not to execute or function properly. Key files will typically be used to validate that a software component referenced by the CoSWID tag document is actually installed on a specific system entity.

location (index 23): The directory or location where a file was found or can expected to be located. This text-string is intended to include the filename itself. This SHOULD be the relative path from the location represented by the root item.

fs-name (index 24): The file name or directory name without any path characters.

root (index 25): A system-specific root folder that the location item is an offset from. If this is not specified the assumption is the root is the same folder as the location of the CoSWID tag. The text-string value represents a path expression relative to the CoSWID tag document location in the (composite) file-system hierarchy.

path-elements (index 26): Provides the ability to apply a directory structure to the path expressions for files defined in a payload or evidence item.

process-name (index 27): The process name as it will be found in the system entity’s process table.

pid (index 28): The process ID for the process in execution that can be included in the process item as part of an evidence tag.

type (index 29): The type of resource represented via a text-string (typically, registry-key, port or root-uri).

2.7.3. The payload Object

The CDDL for the payload object is as follows:

```
payload = {
   global-attributes,
   resource-collection,
   * $$payload-extension
}
```

The following describes each child item of this object.
2.7.4. The evidence Object

The CDDL for the evidence object is as follows:

evidence = {
  global-attributes,
  resource-collection,
  ? date,
  ? device-id,
  * $evidence-extension
}

date = (35: time)
device-id = (36: text)

The following describes each child item of this object.

- global-attributes: The global-attributes group described in Section 2.2.
- resource-collection: The resource-collection group described in Section 2.7.2.
- date (index 35): The date and time evidence represented by an evidence item was gathered.
- device-id (index 36): A text-string identifier for a device evidence was gathered from.
- $evidence-extension: This CDDL socket (see [I-D.ietf-cbor-cddl] section 3.9) can be used to extend the evidence model, allowing well-formed extensions to be defined in additional CDDL descriptions.
2.8. Full CDDL Definition

In order to create a valid CoSWID document the structure of the corresponding CBOR message MUST adhere to the following CDDL data definition.

```cddl
concise-software-identity = {
  global-attributes,
  tag-id,
  tag-version,
  ? corpus,
  ? patch,
  ? supplemental,
  swid-name,
  ? software-version,
  ? version-scheme,
  ? media,
  ? software-meta-entry,
  entity-entry,
  ? link-entry,
  ? ( payload-entry // evidence-entry ),
  ? any-element-entry,
}
```

```cddl
any-uri = text
label = text / int
```

```cddl
any-attribute = (
  label => text / int / [ 2* text ] / [ 2* int ]
)
```

```cddl
any-element-map = {
  global-attributes,
  * label => any-element-map / [ 2* any-element-map ],
}
```

```cddl
global-attributes = (?
  lang,
  * any-attribute,
)
```

```cddl
resource-collection = (?
  directory-entry,
  ? file-entry,
  ? process-entry,
  ? resource-entry
)
```
file = {
    filesystem-item,
    ? size,
    ? file-version,
    ? hash-entry,
}
filesystem-item = {
    global-attributes,
    ? key,
    ? location,
    fs-name,
    ? root,
}
directory = {
    filesystem-item,
    path-elements,
}
process = {
    global-attributes,
    process-name,
    ? pid,
}
resource = {
    global-attributes,
    type,
}
entity = {
    global-attributes,
    entity-name,
    ? reg-id,
    role,
    ? thumbprint,
    extended-data,
}
evidence = {
    global-attributes,
    resource-collection,
    ? date,
    ? device-id,
    * $$evidence-extension
}
link = {
    global-attributes,
    ? artifact,
    href,
    ? media
    ? ownership,
    rel,
    ? media-type,
    ? use,
}

software-meta = {
    global-attributes,
    ? activation-status,
    ? channel-type,
    ? colloquial-version,
    ? description,
    ? edition,
    ? entitlement-data-required,
    ? entitlement-key,
    ? generator,
    ? persistent-id,
    ? product,
    ? product-family,
    ? revision,
    ? summary,
    ? unspsc-code,
    ? unspsc-version,
}

payload = {
    global-attributes,
    resource-collection,
    * $$payload-extension
}

tag-id = (0: text)
swid-name = (1: text)
entity-entry = (2: entity / [ 2* entity ])
evidence-entry = (3: evidence)
link-entry = (4: link / [ 2* link ])
software-meta-entry = (5: software-meta / [ 2* software-meta ])
payload-entry = (6: payload)
any-element-entry = (7: any-element-map / [ 2* any-element-map ])
corpus = (8: bool)
patch = (9: bool)
media = (10: [ + [ media-expression,
               ? [ media-operation,
media-operation = text
media-expression = media-environment / [ media-prefix,
  media-environment,
  media-attribute,
  media-value,
]

media-prefix = text
media-environment = text
media-attribute = text
media-value = text
supplemental = (11: bool)
tag-version = (12: integer)
software-version = (13: text)
version-scheme = (14: text / int)
lang = (15: text)
directory-entry = (16: directory / [ 2* directory ])
file-entry = (17: file / [ 2* file ])
process-entry = (18: process / [ 2* process ])
resource-entry = (19: resource / [ 2* resource ])
size = (20: integer)
file-version = (21: text)
key = (22: bool)
location = (23: text)
fs-name = (24: text)
root = (25: text)
path-elements = (26: { * file-entry,
  * directory-entry,
}

process-name = (27: text)
pid = (28: integer)
type = (29: text)
extended-data = (30: any-element-map / [ 2* any-element-map ])
entity-name = (31: text)
reg-id = (32: any-uri)
role = (33: roles / [ 2* roles ] / text / [ 2* text ])
roles= aggregator / distributor / licensor / software-creator / tag-creator
aggregator=0
distributor=1
licensor=2
software-creator=3
tag-creator=4
thumbprint = (34: [ hash-alg-id: int,
  hash-value: bstr,
3. CoSWID Indexed Label Values

3.1. Version Scheme

The following are an initial set of values for use in the version-scheme item for the version schemes defined in the ISO/IEC 19770-2:2015 [SWID] specification. Index value in parens indicates the index value to use in the version-scheme item.

- multipartnumeric (index 1): Numbers separated by dots, where the numbers are interpreted as integers (e.g., 1.2.3, 1.4.5, 1.2.3.4.5.6.7)
- multipartnumeric+suffix (index 2): Numbers separated by dots, where the numbers are interpreted as integers with an additional string suffix (e.g., 1.2.3a)
- alphanumeric (index 3): Strictly a string, sorting is done alphanumerically
- decimal (index 4): A floating point number (e.g., 1.25 is less than 1.3)
- semver (index 16384): Follows the [SEMVER] specification

The values above are registered in the "SWID/CoSWID Version Schema Values" registry defined in Section 4.1. Additional valid values will likely be registered over time in this registry.

3.2. Entity Role Values

The following table indicates the index value to use for the entity roles defined in the ISO/IEC 19770-2:2015 [SWID] specification.
### Table

<table>
<thead>
<tr>
<th>Index</th>
<th>Role Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Reserved</td>
</tr>
<tr>
<td>1</td>
<td>tagCreator</td>
</tr>
<tr>
<td>2</td>
<td>softwareCreator</td>
</tr>
<tr>
<td>3</td>
<td>aggregator</td>
</tr>
<tr>
<td>4</td>
<td>distributor</td>
</tr>
<tr>
<td>5</td>
<td>licensor</td>
</tr>
</tbody>
</table>

The values above are registered in the "SWID/CoSWID Entity Role Values" registry defined in section Section 4.2. Additional valid values will likely be registered over time. Additionally, the index values 226 through 255 have been reserved for private use.

### 4. IANA Considerations

This document will include requests to IANA:

- Integer indices for SWID content attributes and information elements.
- Content-Type for CoAP to be used in COSE.

This document has a number of IANA considerations, as described in the following subsections.

#### 4.1. SWID/CoSWID Version Schema Values Registry

This document uses unsigned 16-bit index values to version-scheme item values. The initial set of version-scheme values are derived from the textual version scheme names defined in the ISO/IEC 19770-2:2015 specification [SWID].

This document defines a new registry entitled "SWID/CoSWID Version Schema Values". Future registrations for this registry are to be made based on [RFC8126] as follows:
Initial registrations for the SWID/CoSWID Version Schema Values registry are provided below.

<table>
<thead>
<tr>
<th>Index</th>
<th>Role Name</th>
<th>Specification</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>multipartnumeric</td>
<td>See Section 3.1</td>
</tr>
<tr>
<td>2</td>
<td>multipartnumeric+suffix</td>
<td>See Section 3.1</td>
</tr>
<tr>
<td>3</td>
<td>alphanumeric</td>
<td>See Section 3.1</td>
</tr>
<tr>
<td>4</td>
<td>decimal</td>
<td>See Section 3.1</td>
</tr>
<tr>
<td>5-16383</td>
<td>Unassigned</td>
<td></td>
</tr>
<tr>
<td>16384</td>
<td>semver</td>
<td>[SEMVER]</td>
</tr>
<tr>
<td>16385-32767</td>
<td>Unassigned</td>
<td></td>
</tr>
<tr>
<td>32768-65535</td>
<td>Reserved for Private Use</td>
<td></td>
</tr>
</tbody>
</table>

4.2. SWID/CoSWID Entity Role Values Registry

This document uses unsigned 8-bit index values to represent entity-role values. The initial set of Entity roles are derived from the textual role names defined in the ISO/IEC 19770-2:2015 specification [SWID].

This document defines a new registry entitled "SWID/CoSWID Entity Role Values". Future registrations for this registry are to be made based on [RFC8126] as follows:
Initial registrations for the SWID/CoSWID Entity Role Values registry are provided below.

<table>
<thead>
<tr>
<th>Index</th>
<th>Role Name</th>
<th>Specification</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>tagCreator</td>
<td>See Section 3.2</td>
</tr>
<tr>
<td>2</td>
<td>softwareCreator</td>
<td>See Section 3.2</td>
</tr>
<tr>
<td>3</td>
<td>aggregator</td>
<td>See Section 3.2</td>
</tr>
<tr>
<td>4</td>
<td>distributor</td>
<td>See Section 3.2</td>
</tr>
<tr>
<td>5</td>
<td>licensor</td>
<td>See Section 3.2</td>
</tr>
<tr>
<td>6-49</td>
<td>Unassigned</td>
<td></td>
</tr>
<tr>
<td>50-225</td>
<td>Unassigned</td>
<td></td>
</tr>
<tr>
<td>225-255</td>
<td>Reserved for Private Use</td>
<td></td>
</tr>
</tbody>
</table>

5. Security Considerations

SWID and CoSWID tags contain public information about software components and, as such, do not need to be protected against disclosure on an endpoint. Similarly, SWID tags are intended to be easily discoverable by applications and users on an endpoint in order to make it easy to identify and collect all of an endpoint’s SWID tags. As such, any security considerations regarding SWID tags focus on the application of SWID tags to address security challenges, and the possible disclosure of the results of those applications.

A signed SWID tag whose signature has been validated can be relied upon to be unchanged since it was signed. If the SWID tag was
created by the software provider, is signed, and the software provider can be authenticated as the originator of the signature, then the tag can be considered authoritative. In this way, an authoritative SWID tag contains information about a software product provided by the maintainer of the product, who is expected to be an expert in their own product. Thus, authoritative SWID tags can be trusted to represent authoritative information about the software product. Having an authoritative SWID tag can be useful when the information in the tag needs to be trusted, such as when the tag is being used to convey reference integrity measurements for software components. By contrast, the data contained in unsigned tags cannot be trusted to be unmodified.

SWID tags are designed to be easily added and removed from an endpoint along with the installation or removal of software components. On endpoints where addition or removal of software components is tightly controlled, the addition or removal of SWID tags can be similarly controlled. On more open systems, where many users can manage the software inventory, SWID tags may be easier to add or remove. On such systems, it may be possible to add or remove SWID tags in a way that does not reflect the actual presence or absence of corresponding software components. Similarly, not all software products automatically install SWID tags, so products may be present on an endpoint without providing a corresponding SWID tag. As such, any collection of SWID tags cannot automatically be assumed to represent either a complete or fully accurate representation of the software inventory of the endpoint. However, especially on devices that more strictly control the ability to add or remove applications, SWID tags are an easy way to provide an preliminary understanding of that endpoint’s software inventory.

Any report of an endpoint’s SWID tag collection provides information about the software inventory of that endpoint. If such a report is exposed to an attacker, this can tell them which software products and versions thereof are present on the endpoint. By examining this list, the attacker might learn of the presence of applications that are vulnerable to certain types of attacks. As noted earlier, SWID tags are designed to be easily discoverable by an endpoint, but this does not present a significant risk since an attacker would already need to have access to the endpoint to view that information. However, when the endpoint transmits its software inventory to another party, or that inventory is stored on a server for later analysis, this can potentially expose this information to attackers who do not yet have access to the endpoint. As such, it is important to protect the confidentiality of SWID tag information that has been collected from an endpoint, not because those tags individually contain sensitive information, but because the collection of SWID
tags and their association with an endpoint reveals information about that endpoint’s attack surface.

Finally, both the ISO-19770-2:2015 XML schema definition and the Concise SWID data definition allow for the construction of "infinite" SWID tags or SWID tags that contain malicious content with the intent if creating non-deterministic states during validation or processing of SWID tags. While software product vendors are unlikely to do this, SWID tags can be created by any party and the SWID tags collected from an endpoint could contain a mixture of vendor and non-vendor created tags. For this reason, tools that consume SWID tags ought to treat the tag contents as potentially malicious and should employ input sanitizing on the tags they ingest.

6. Acknowledgments

7. Change Log

Changes from version 05 to version 06:
- Improved quantities
- Included proposals for implicit enumerations that were NMTOKENS
- Added extension points
- Improved exemplary firmware-resource extension

Changes from version 04 to version 05:
- Clarified language around SWID and CoSWID to make more consistent use of these terms.
- Added language describing CBOR optimizations for single vs. arrays in the model front matter.
- Fixed a number of grammatical, spelling, and wording issues.
- Documented extension points that use CDDL sockets.
- Converted IANA registration tables to markdown tables, reserving the 0 value for use when a value is not known.
- Updated a number of references to their current versions.

Changes from version 03 to version 04:
- Re-index label values in the CDDL.
Added a section describing the CoSWID model in detail.

Created IANA registries for entity-role and version-scheme

Changes from version 02 to version 03:

- Updated CDDL to allow for a choice between a payload or evidence
- Re-index label values in the CDDL.
- Added item definitions
- Updated references for COSE, CBOR Web Token, and CDDL.

Changes from version 01 to version 02:

- Added extensions for Firmware and CoSWID use as Reference Integrity Measurements (CoSWID RIM)
- Changes meta handling in CDDL from use of an explicit use of items to a more flexible unconstrained collection of items.
- Added sections discussing use of COSE Signatures and CBOR Web Tokens

Changes from version 00 to version 01:

- Added CWT usage for absolute SWID paths on a device
- Fixed cardinality of type-choices including arrays
- Included first iteration of firmware resource-collection

Changes since adopted as a WG I-D -00:

- Removed redundant any-attributes originating from the ISO-19770-2:2015 XML schema definition
- Fixed broken multi-map members
- Introduced a more restrictive item (any-element-map) to represent custom maps, increased restriction on types for the any-attribute, accordingly
- Fixed X.1520 reference
- Minor type changes of some attributes (e.g. NMTOKENS)
Changes from version 00 to version 01:

- Added semantic differentiation of various name types (e.g. fs-name)

Changes from version 01 to version 02:

- Enforced a more strict separation between the core CoSWID definition and additional usage by moving content to corresponding appendices.
- Removed artifacts inherited from the reference schema provided by ISO (e.g. NMTOKEN(S))
- Simplified the core data definition by removing group and type choices where possible
- Minor reordering of map members
- Added a first extension point to address requested flexibility for extensions beyond the any-element

8. Contributors

9. References

9.1. Normative References

[I-D.ietf-ace-cbor-web-token]

9.2. Informative References

[I-D.birkholz-tuda]
Fuchs, A., Birkholz, H., McDonald, I., and C. Bormann,
"Time-Based Uni-Directional Attestation", draft-birkholz-
tuda-04 (work in progress), March 2017.
Appendix A. CoSWID Attributes for Firmware (label 60)

The ISO-19770-2:2015 specification of SWID tags assumes the existence of a file system a software component is installed and stored in. In the case of constrained-node networks [RFC7228] or network equipment this assumption might not apply. Concise software instances in the form of (modular) firmware are often stored directly on a block device that is a hardware component of the constrained-node or network equipment. Multiple differentiable block devices or segmented block devices that contain parts of modular firmware components (potentially each with their own instance version) are already common at the time of this writing.

The optional attributes that annotate a firmware package address specific characteristics of pieces of firmware stored directly on a block-device in contrast to software deployed in a file-system. In essence, trees of relative path-elements expressed by the directory
and file structure in CoSWID tags are typically unable to represent the location of a firmware on a constrained-node (small thing). The composite nature of firmware and also the actual composition of small things require a set of attributes to address the identification of the correct component in a composite thing for each individual piece of firmware. A single component also potentially requires a number of distinct firmware parts that might depend on each other (versions). These dependencies can be limited to the scope of the component itself or extend to the scope of a larger composite device. In addition, it might not be possible (or feasible) to store a CoSWID tag document (permanently) on a small thing along with the corresponding piece of firmware.

To address the specific characteristics of firmware, the extension points "$\$payload-extension" and "$\$evidence-extension" are used to allow for an additional type of resource description--firmware-entry--thereby increasing the self-descriptiveness and flexibility of CoSWID. The optional use of the extension points "$\$payload-extension" and "$\$evidence-extension" in respect to firmware MUST adhere to the following CDDL data definition.

```cddl
<CODE BEGINS>

$$payload-extension  //= (firmware-entry,)
$$evidence-extension  //= (firmware-entry,)

firmware-manifest = {
  firmware-manifest-id,
  firmware-manifest-creation-timestamp,
  firmware-manifest-description,
  firmware-manifest-version,
  firmware-manifest-nonc,
  ? firmware-manifest-aliases,
  ? firmware-manifest-dependencies,
  firmware-target-device-identifier,
  firmware-payload-entry,
  ? simple-firmware-manifest-extensions,  //
  $$firmware-manifest-extensions,
}

firmware-payload = {
  firmware-payload-id,
  ? firmware-package-identifier,
  firmware-payload-description,
  firmware-payload-format,
  firmware-payload-size,
  ? firmware-payload-simple-version,
  ? firmware-payload-version,
  firmware-payload-digests,
}

<CODE ENDS>
```


firmware-entry = (59: firmware-manifest / [ 2* firmware-manifest ])
firmware-payload-entry = (60: firmware-payload / [ 2* firmware-payload ])
firmware-payload-id = (61: bytes / text / uint)
firmware-package-identifier = (62: text)
firmware-manifest-id = (63: bytes / text / int)
firmware-manifest-creation-timestamp = (64: time)
firmware-manifest-version = (65: uint)
firmware-manifest-description = (66: text)
firmware-manifest-nonc = (67: bytes)
firmware-manifest-dependencies = (68: resource-reference)
firmware-manifest-aliases = (69: resource-reference)
resource-reference = [ + [ resource-reference-uri: uri,
   resource-reference-digest: bytes,
   ]],

firmware-payload-description = (70: text)
firmware-payload-format = (71: { firmware-payload-format-type,
   ? firmware-payload-format-guidance,
   })
firmware-payload-format-type = (72: int)
firmware-payload-format-guidance = (73: bytes)
firmware-payload-size = (74: uint)
firmware-payload-package = (75: { ? firmware-package-compression-type,
   ? firmware-package-compression-guidance,
   firmware-package,
   })
firmware-package-compression-type = (76: text / int)
firmware-package-compression-guidance = (77: bytes)
firmware-package = (78: bytes)
firmware-target-component-index = (79: text)
firmware-target-storage-identifier = (80: bytes / text / int)
firmware-target-dependency = (81: [ ? | firmware-target-major-version,
   version-comparison,
   required-version,
   ])

firmware-payload-relationships = (82: [ + { firmware-payload-relationship-type,
            firmware-payload-ids,
           }],
           firmware-payload-relationship-types)
firmware-payload-relationship-type = (84: $firmware-payload-relationship-types)
$firmware-payload-relationship-types /= patches-firmware
$firmware-payload-relationship-types /= requires-firmware
$firmware-payload-relationship-types /= supersedes-firmware
patches-firmware = 1
requires-firmware = 2
supersedes-firmware = 3
firmware-target-device-identifier = (85: { firmware-target-vendor-identifier,
                                   ? firmware-target-type-identifier,
                                   firmware-target-model-identifier,
                                   ? firmware-target-class-identifier,
                                   ? firmware-target-rfc4122-identifier,
                                   ? firmware-target-8021AR-identifier,
                                   $$firmware-target-identifier-extensions,
                                 }
                                 firmware-target-vendor-identifier = (86: text)
firmware-target-type-identifier = (87: text)
firmware-target-model-identifier = (88: text)
firmware-target-class-identifier = (89: text)
firmware-target-rfc4122-identifier = (90: text)
firmware-target-8021AR-identifier = (91: bytes)
firmware-target-minimal-version  = (92: { firmware-target-major-version,
                                           firmware-target-minor-version,
                                           ? firmware-target-revision-version,
                                           ? firmware-target-build-version,
                                           ? firmware-target-storage-identifier,
firmware-target-major-version = (93: uint)
firmware-target-minor-version = (94: uint)
firmware-target-revision-version = (95: uint)
firmware-target-build-version = (96: uint)
firmware-payload-digests = (97: [ + { firmware-digest-type,
   ? firmware-digest-config-guidance,
   firmware-digest,
   }],
)
firmware-digest-type = (98: $firmware-digest-types)
$firmware-digest-types /= raw-payload-digest
$firmware-digest-types /= installed-payload-digest
$firmware-digest-types /= ciphertext-digest
$firmware-digest-types /= pre-image-digest
raw-payload-digest = 1
installed-payload-digest = 2
ciphertext-digest = 3
pre-image-digest = 4
firmware-digest-config-guidance = (99: bytes)
firmware-digest = (100: bytes)
firmware-payload-conditions = (101: [ + { firmware-payload-condition-type,
   firmware-payload-condition-parameters,
   },
   ]
)
firmware-payload-condition-parameters = (102: bytes)
firmware-payload-condition-type = (103: $firmware-payload-condition-types)
$firmware-payload-condition-types /= vendor-id-condition
$firmware-payload-condition-types /= class-id-condition
$firmware-payload-condition-types /= device-id-condition
$firmware-payload-condition-types /= best-before-condition
vendor-id-condition = 1
class-id-condition = 2
device-id-condition = 3
best-before-condition = 4
firmware-payload-directives = (104: [ + { firmware-payload-directive-type,
   firmware-payload-directive-parameters,
   },
   ]
)
firmware-payload-directive-parameters = (105: bytes)
firmware-payload-directive-type = (106: $firmware-payload-directive-types)
$firmware-payload-directive-types /= apply-immediately-directive
$firmware-payload-directive-types /= apply-after-directive
apply-immediately-directive = 1
apply-after-directive = 2
firmware-payload-simple-version = (107: uint)
firmware-payload-version = (108: { firmware-payload-major-version,
    firmware-payload-minor-version,
    ? firmware-payload-revision-version,
    ? firmware-payload-build-version,
    }
    firmware-payload-major-version = (109: uint)
firmware-payload-minor-version = (110: uint)
firmware-payload-revision-version = (111: uint)
firmware-payload-build-version = (112: uint)
version-comparison = (113: eq / ne / lt / le / gt / ge)
required-version = (114: uint)
simple-firmware-manifest-extensions = (115: { + int => bytes })
simple-firmware-payload-extensions = (116: { + int => bytes })

\textbf{The members of the firmware group that constitutes the content of the firmware-entry is based on the metadata about firmware Described in [RFC4108]. As with every semantic differentiation that is supported by the resource-collection type, the use of firmware-entry is optional. It is REQUIRED not to instantiate more than one firmware-entry, as the firmware group is used in a map and therefore only allows for unique labels.}

The optional cms-firmware-package member allows to include the actual firmware in the CoSWID tag that also expresses its metadata as a byte-string. This option enables a CoSWID tag to be used as a container or wrapper that composes both firmware and its metadata in a single document (which again can be signed, encrypted and/or compressed). In consequence, a CoSWID tag about firmware can be conveyed as an identifying document across endpoints or used as a reference integrity measurement as usual. Alternatively, it can also convey an actual piece of firmware, serve its intended purpose as a SWID tag and then - due to the lack of a location to store it - be discarded.
Appendix B. Signed Concise SWID Tags using COSE

SWID tags, as defined in the ISO-19770-2:2015 XML schema, can include cryptographic signatures to protect the integrity of the SWID tag. In general, tags are signed by the tag creator (typically, although not exclusively, the vendor of the software component that the SWID tag identifies). Cryptographic signatures can make any modification of the tag detectable, which is especially important if the integrity of the tag is important, such as when the tag is providing reference integrity measurements for files.

The ISO-19770-2:2015 XML schema uses XML DSIG to support cryptographic signatures. CoSWID tags require a different signature scheme than this. COSE (CBOR Object Signing and Encryption) provides the required mechanism [RFC8152]. Concise SWID can be wrapped in a COSE Single Signer Data Object (cose-sign1) that contains a single signature. The following CDDL defines a more restrictive subset of header attributes allowed by COSE tailored to suit the requirements of Concise SWID.

<CODE BEGINS>
signed-coswid = #6.997(COSE-Sign1-coswid) ; see TBS7 in current COSE I-D

label = int / tstr ; see COSE I-D 1.4.
values = any ; see COSE I-D 1.4.

unprotected-signed-coswid-header = {
  1 => int, ; algorithm identifier
  3 => "application/coswid", ; request for CoAP IANA registry to become an int
    * label => values,
}

protected-signed-coswid-header = {
  4 => bstr, ; key identifier
    * label => values,
}

COSE-Sign1-coswid = [
  protected: bstr .cbor protected-signed-coswid-header,
  unprotected: unprotected-signed-coswid-header,
  payload: bstr .cbor concise-software-identity,
  signature: bstr,
]
<CODE ENDS>
Appendix C. CoSWID used as Reference Integrity Measurements (CoSWID RIM)

A vendor supplied signed CoSWID tag that includes hash-values for the files that compose a software component can be used as a RIM (reference integrity measurement). A RIM is a type of declarative guidance that can be used to assert the compliance of an endpoint by assessing the installed software. In the context of remote attestation based on an attestation via hardware rooted trust, a verifier can appraise the integrity of the conveyed measurements of software components using a CoSWID RIM provided by a source, such as [I-D.ietf-sacm-rolie-softwaredescriptor].

RIM Manifests (RIMM): A group of SWID tags about the same (sub-)system, system entity, or (sub-)component (compare [RFC4949]). A RIMM manifest is a distinct document that is typically conveyed en-block and constitutes declarative guidance in respect to a specific (target) endpoint (compare [I-D.ietf-sacm-terminology]).

If multiple CoSWID compose a RIMM, the following CDDL data definition SHOULD be used.

RIMM = [ + concise-software-identity / signed-coswid ]

Appendix D. CBOR Web Token for Concise SWID Tags

A typical requirement regarding specific instantiations of endpoints - and, as a result, specific instantiations of software components - is a representation of the absolute path of a CoSWID tag document in a file system in order to derive absolute paths of files represented in the corresponding CoSWID tag. The absolute path of an evidence CoSWID tag can be included as a claim in the header of a CBOR Web Token [I-D.ietf-ace-cbor-web-token]. Depending on the source of the token, the claim can be in the protected or unprotected header portion.

<CODE BEGINS>
CDDL TBD
<CODE ENDS>

Authors’ Addresses
Internet-Draft                  COSWID                        July 2018

Henk Birkholz
Fraunhofer SIT
Rheinstrasse 75
Darmstadt  64295
Germany

Email: henk.birkholz@sit.fraunhofer.de

Jessica Fitzgerald-McKay
Department of Defense
9800 Savage Road
Ft. Meade, Maryland
USA

Email: jmfitz2@nsa.gov

Charles Schmidt
The MITRE Corporation
202 Burlington Road
Bedford, Maryland  01730
USA

Email: cmschmidt@mitre.org

David Waltermire
National Institute of Standards and Technology
100 Bureau Drive
Gaithersburg, Maryland  20877
USA

Email: david.waltermire@nist.gov
Security Automation and Continuous Monitoring (SACM) Terminology

draft-ietf-sacm-terminology-15

Abstract

This memo documents terminology used in the documents produced by SACM (Security Automation and Continuous Monitoring).

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 December 15, 2018.

Copyright Notice

Copyright (c) 2018 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust’s Legal Provisions Relating to IETF Documents (https://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect
# 1. Introduction

Our goal with this document is to improve our agreement on the terminology used in documents produced by the IETF Working Group for Security Automation and Continuous Monitoring. Agreeing on terminology should help reach consensus on which problems we’re trying to solve, and propose solutions and decide which ones to use.

# 2. Terms and Definitions

This section describes terms that have been defined by other RFC’s and defines new ones. The predefined terms will reference the RFC and where appropriate will be annotated with the specific context by which the term is used in SACM. Note that explanatory or informational augmentation to definitions are segregated from the definitions themselves. The definition for the term immediately follows the term on the same line, whereas expositional text is contained in subsequent paragraphs immediately following the definition.

**Assertion:** Defined by the ITU in [X.1252] as "a statement made by an entity without accompanying evidence of its validity".

In the context of SACM, an assertion is the output of a SACM Component in the form of a SACM Statement (including metadata about the data source and data origin, e.g. timestamps). While the validity of an assertion about Content and Content Metadata cannot be verified without, for example, Integrity Proofing of the...
Data Source, an assertion (and therefore a SACM statement, respectively) of the validity of Statement Metadata can be enabled by including corresponding Integrity Evidence created by the Data Origin.

Assessment: Defined in [RFC5209] as "the process of collecting posture for a set of capabilities on the endpoint (e.g., host-based firewall) such that the appropriate validators may evaluate the posture against compliance policy."

Asset: Is a system resource, as defined in [RFC4949], that may be composed of other assets.

Examples of Assets include: Endpoints, Software, Guidance, or X.509 public key certificates. An asset is not necessarily owned by an organization.

Asset Management: The IT process by which assets are provisioned, updated, maintained and deprecated.

Attribute: Is a data element, as defined in [RFC5209], that is atomic.

In the context of SACM, attributes are "atomic" information elements and an equivalent to attribute-value-pairs. Attributes can be components of Subjects.

Broken remnant of a term again, but this time left here to show how much the last submit of -14 broke the document (this is actually not a term definition, apparently, but if you are curious this was "Authorization", became a second paragraph of expositional text to the definition of Attribute and now became the universal disclaimer of "please alter the structure of the document with care") - until removal by a less annoyed editor:

Defined in [RFC4949] as "an approval that is granted to a system entity to access a system resource."

Capability: A set of features that are available from a SACM Component.

See also "capability" in [I-D.ietf-i2nsf-terminology].

In the context of SACM, the extent of a SACM component’s ability is enabled by the functions it is composed of. Capabilities are registered at a SACM broker (potentially also at a proxy or a repository component if it includes broker functions) by a SACM component via the SACM component registration task and can be discovered by or negotiated with other SACM components via the
corresponding tasks. For example, the capability of a SACM provider may be to provide target endpoint records (declarative guidance about well-known or potential target endpoints), or only a subset of that data.

A capability’s description is in itself imperative guidance on what functions are exposed to other SACM components in a SACM domain and how to use them in workflows.

The SACM Vulnerability Assessment Scenario [I-D.ietf-sacm-vuln-scenario] defines the terms Endpoint Management Capabilities, Vulnerability Management Capabilities, and Vulnerability Assessment Capabilities, which illustrate specific sets of SACM capabilities on an enterprise IT department’s point of view and therefore compose sets of declarative guidance.

Collection Result: Is a composition of one or more content elements carrying information about a target endpoint, that is produced by a collector when conducting a collection task.

Collection Task: A targeted task that collects attributes and/or corresponding attribute values from target endpoint.

There are four types of frequency collection tasks can be conducted with:

- ad-hoc, e.g. triggered by an unsolicited query
- conditional, e.g. triggered in accordance with policies included in the compositions of workflows
- scheduled, e.g. in regular intervals, such as every minute or weekly
- continuously, e.g. a network behavior observation

There are three types of collection methods, each requiring an appropriate set of functions to be included in the SACM component conducting the collection task:

- Self-Reporting: A SACM component located on the target endpoint itself conducts the collection task.
- Remote-Acquisition: A SACM component located on an Endpoint different from the target endpoint conducts the collection task via interfaces available on the target endpoint, e.g. SNMP/NETCONF or WMI.
Behavior-Observation: A SACM component located on an Endpoint different from the target endpoint observes network traffic related to the target endpoint and conducts the collection task via interpretation of that network traffic.

Collector: A piece of software that acquires information about one or more target endpoints by conducting collection tasks.

A collector can be distributed across multiple endpoints, e.g. across a target endpoint and a SACM component. The separate parts of the collector can communicate with a specialized protocol, such as PA-TNC [RFC5792]. At least one part of a distributed collector has to take on the role of a provider of information by providing SACM interfaces to propagate capabilities and to provide SACM content in the form of collection results.

Configuration: A non-volatile subset of the endpoint attributes of an endpoint that is intended to be unaffected by a normal reboot-cycle.

Configuration is a type of imperative guidance that is stored in files (files dedicated to contain configuration and/or files that are software components), directly on block devices, or on specific hardware components that can be accessed via corresponding software components. Modification of configuration can be conducted manually or automatically via management (plane) interfaces that support management protocols, such as SNMP or WMI. A change of configuration can occur during both run-time and down-time of an endpoint. It is common practice to scheduled a change of configuration during or directly after the completion of a boot-cycle via corresponding software components located on the target endpoint itself.

Examples: The static association of an IP address and a MAC address in a DHCP server configuration, a directory-path that identifies a log-file directory, a registry entry.

Configuration Drift: The disposition of endpoint characteristics to change over time.

Configuration drift exists for both hardware components and software components. Typically, the frequency and scale of configuration drift of software components is significantly higher than the configuration drift of hardware components.

Consumer: A SACM Role that requires a SACM Component to include SACM Functions enabling it to receive information from other SACM Components.
Content Element: Content elements constitute the payload data (SACM content) transferred via statement Subjects emitted by providers of information. Every content element Subject includes a specific content Subject and a corresponding content metadata Subject.

Content Metadata: Data about content Subjects. Every content-element includes a content metadata Subject. The Subject can include any information element that can annotate the content transferred. Examples include time stamps or data provenance Subjects.

Control Plane: An architectural component that provides common control functions to all SACM components.

Typically used as a term in the context of routing, e.g. [RFC6192]. SACM components may include authentication, authorization, (capability) discovery or negotiation, registration and subscription. The control plane orchestrates the flow on the data plane according to imperative guidance (i.e. configuration) received via the management plane. SACM components with interfaces to the control plane have knowledge of the capabilities of other SACM components within a SACM domain.

Controller: A controller is a SACM Role that is assigned to a SACM component containing control plane functions managing and facilitating information sharing or execute on security functions.

There are three types of SACM controllers: Broker, Proxy, and Repository. Depending on its type, a controller can also contain functions that have interfaces on the data plane.

Data Confidentiality: Defined in [RFC4949] as "the property that data is not disclosed to system entities unless they have been authorized to know the data."

Data In Motion: Data that is being transported via a network; also referred to as "Data in Transit" or "Data in Flight".

Data in motion requires a data model to transfer the data using a specific encoding. Typically, data in motion is serialized (marshalling) into a transport encoding by a provider of information and deserialized (unmarshalling) by a consumer of information. The termination points of provider of information and consumer of information data is transferred between are interfaces. In regard to data in motion, the interpretation of the roles consumer of information and provider of information depends on the corresponding OSI layer (e.g. on layer2: between interfaces connected to a broadcast domain, on layer4: between
interfaces that maintain a TCP connection). In the context of SACM, consumer of information and provider of information are SACM components.

Data At Rest: Data that is stored.

Data at rest requires a data model to encode the data to be stored. In the context of SACM, data at rest located on a SACM component can be provided to other SACM components via discoverable capabilities.

Data Integrity: Defined in [RFC4949] as "the property that data has not been changed, destroyed, or lost in an unauthorized or accidental manner."

Data Origin: The SACM Component that initially acquired or produced data about an endpoint.

Data Origin enables a SACM component to identify the SACM component that initially acquired or produced data about a (target) endpoint (e.g. via collection from a data source) and made it available to a SACM domain via a SACM statement. Data Origin can be expressed by an endpoint label information element (e.g. to be used as metadata in statement).

Data Plane: Is an architectural component providing operational functions enabling information exchange that is not command and control or management related.

Typically used as a term in the context of routing (and used as a synonym for forwarding plane, e.g. [RFC6192]). In the context of SACM, the data plane is an architectural component providing operational functions to enable a SACM component to provide and consume SACM statements and therefore SACM content, which composes the actual SACM content. The data plane in a SACM domain is used to conduct distributed SACM tasks by transporting SACM content via specific transport encodings and corresponding operations defined by SACM data models.

Data Provenance: An historical record of the sources, origins and evolution, as it pertains to data, that is influenced by inputs, entities, functions and processes.

Additional Information - In the context of SACM, data provenance is expressed as metadata that identifies SACM statements and corresponding content elements a new statement is created from. In a downstream process, this references can cascade, creating a data provenance tree that enables SACM components to trace back
the original data sources involved in the creation of SACM statements and take into account their characteristics and trustworthiness.

Data Source: Is an endpoint from which a particular set of attributes and/or attribute values have been collected.

Data Source enables a SACM component to identify - and potentially characterize - a (target) endpoint that is claimed to be the original source of endpoint attributes in a SACM statement. Data Source can be expressed as metadata by an endpoint label information element or a corresponding subject of identifying endpoint attributes.

Endpoint: Defined in [RFC5209] as "any computing device that can be connected to a network."

Additional Information - The [RFC5209] definition continues, "Such devices normally are associated with a particular link layer address before joining the network and potentially an IP address once on the network. This includes: laptops, desktops, servers, cell phones, or any device that may have an IP address."

To further clarify the [RFC5209] definition, an endpoint is any physical or virtual device that may have a network address. Note that, network infrastructure devices (e.g. switches, routers, firewalls), which fit the definition, are also considered to be endpoints within this document.

Physical endpoints are always composites that are composed of hardware components and software components. Virtual endpoints are composed entirely of software components and rely on software components that provide functions equivalent to hardware components.

The SACM architecture differentiates two essential categories of endpoints: Endpoints whose security posture is intended to be assessed (target endpoints) and endpoints that are specifically excluded from endpoint posture assessment (excluded endpoints).

Based on the definition of an asset, an endpoint is a type of asset.

Endpoint Attribute: Is a discreet endpoint characteristic that is computably observable.
Endpoint Attributes typically constitute Attributes that can be bundled into Subject (e.g. information about a specific network interface can be represented via a set of multiple AVP).

Endpoint Characteristics: The state, configuration and composition of the software components and (virtual) hardware components a target endpoint is composed of, including observable behavior, e.g. sys-calls, log-files, or PDU emission on a network.

In SACM work-flows, (Target) Endpoint Characteristics are represented via Information Elements.

Endpoint Characterization Task: The task of endpoint characterization that uses endpoint attributes that represent distinct endpoint characteristics.

Endpoint Classification: The categorization of the endpoint into one or more taxonomic structures.

Endpoint classification requires declarative guidance in the form of an endpoint profile, discovery results and potentially collection results. Types, classes or the characteristics of an individual target endpoint are defined via endpoint profiles.

Endpoint Classification Task: The task of endpoint classification that uses an endpoint’s characteristics to determine how to categorize the given endpoint into one or more taxonomic structures.

Endpoint Label: A unique label associated with a unique endpoint.

Endpoint specializations have corresponding endpoint label specializations. For example, an endpoint label used on a SACM Component is a SACM Component Label.

Endpoint Management Capabilities: Enterprise IT management capabilities that are tailored to manage endpoint identity, endpoint information, and associated metadata.

Evaluation Task: A task by which an endpoint’s asserted attribute value is evaluated against a policy-compliant attribute value.

Evaluation Result: The resulting value from having evaluated a set of posture attributes.

Expected Endpoint Attribute State: The policy-compliant state of an endpoint attribute that is to be compared against.
Sets of expected endpoint attribute states are transported as declarative guidance in target endpoint profiles via the management plane. This, for example, can be a policy, but also a recorded past state. An expected state is represented by an Attribute or a Subject that represents a set of multiple attribute value pairs.

Guidance: Machine-processable input directing SACM processes or tasks.

Examples of such processes/tasks include automated device management, remediation, collection, evaluation. Guidance influences the behavior of a SACM Component and is considered content of the management plane. In the context of SACM, guidance is machine-readable and can be manually or automatically generated or provided. Typically, the tasks that provide guidance to SACM components have a low-frequency and tend to be sporadic.

There are two types of guidance:

Declarative Guidance: Guidance that defines the configuration or state an endpoint is supposed to be in, without providing specific actions or methods to produce that desired state. Examples include Target Endpoint Profiles or network topology based requirements.

Imperative Guidance: Guidance that prescribes specific actions to be conducted or methods to be used in order to achieve an outcome. Examples include a targeted Collection Task or the IP-Address of a SACM Component that provides a registration function.

Prominent examples include: modification of the configuration of a SACM component or updating a target endpoint profile that resides on an evaluator. In essence, guidance is transported via the management plane.

Endpoint Hardware Inventory: The set of hardware components that compose a specific endpoint representing its hardware configuration.

Hardware Component: A distinguishable physical component used to compose an endpoint.

The composition of an endpoint can be changed over time by adding or removing hardware components. In essence, every physical endpoint is potentially a composite of multiple hardware components, typically resulting in a hierarchical composition of hardware components. The composition of hardware components is
based on interconnects provided by specific hardware types (e.g. FRU in a chassis are connected via redundant busses). In general, a hardware component can be distinguished by its serial number. Occasionally, hardware components are referred to as power sucking aliens.

Information Element: A representation of information about physical and virtual "objects of interest".

Information elements are the building blocks that constitute the SACM information model. In the context of SACM, an information element that expresses a single value with a specific name is referred to as an Attribute (analogous to an attribute-value-pair). A set of attributes that is bundled into a more complex composite information element is referred to as a Subject. Every information element in the SACM information model has a unique name. Endpoint attributes or time stamps, for example, are represented as information elements in the SACM information model.

Information Model: An abstract representation of data, their properties, relationships between data and the operations that can be performed on the data.

While there is some overlap with a data model, [RFC3444] distinguishes an information model as being protocol and implementation neutral whereas a data model would provide such details. The purpose of the SACM information model is to ensure interoperability between SACM data models (that are used as transport encoding) and to provide a standardized set of information elements for communication between SACM components.

Interaction Model: The definition of specific sequences regarding the exchange of messages (data in motion), including, for example, conditional branching, thresholds and timers.

An interaction model, for example, can be used to define operations, such as registration or discovery, on the control plane. A composition of data models for data in motion and a corresponding interaction model is a protocol.

Internal Collector: A collector that runs on a target endpoint to acquire information from that target endpoint.

Management Plane: An architectural component providing common functions to steer the behavior of SACM components, e.g. their behavior on the control plane.
Typically, a SACM component can fulfill its purpose without continuous input from the management plane. In contrast, without continuous availability of control plane functions a typical SACM component could not function properly. In general, interaction on the management plane is less frequent and less regular than on the control plane. Input via the management plane can be manual (e.g. via a CLI), or can be automated via management plane functions that are part of other SACM components.

Network Address: A layer-specific address that follows a layer-specific address scheme.

The following characteristics are a summary derived from the Common Information Model and ITU-T X.213. Each Network Interface of a specific layer can be associated with one or more addresses appropriate for that layer. There is no guarantee that a network address is globally unique. A dedicated authority entity can provide a level of assurance that a network address is unique in its given scope. In essence, there is always a scope to a network address, in which it is intended to be unique.

Examples include: physical Ethernet port with a MAC address, layer 2 VLAN interface with a MAC address, layer 3 interface with multiple IPv6 addresses, layer 3 tunnel ingress or egress with an IPv4 address.

Network Interface: An Endpoint is connected to a network via one or more Network Interfaces. Network Interfaces can be physical (Hardware Component) or logical (virtual Hardware component, i.e. a dedicated Software Component). Network Interfaces of an Endpoint can operate on different layers, most prominently what is now commonly called layer 2 and 3. Within a layer, interfaces can be nested.

In SACM, the association of Endpoints and Network Addresses via Network Interfaces is vital to maintain interdependent autonomous processes that can be targeted at Target Endpoints, unambiguously.

Examples include: physical Ethernet port, layer 2 VLAN interface, a MC-LAG setup, layer 3 Point-to-Point tunnel ingress or egress.

Metadata: Data about data.

In the SACM information model, data is referred to as Content. Metadata about the content is referred to as Content-Metadata, respectively. Content and Content-Metadata are combined into Subjects called Content-Elements in the SACM information model. Some information elements defined by the SACM information model...
can be part of the Content or the Content-Metadata. Therefore, if an information element is considered data or data about data depends on which kind of Subject it is associated with. The SACM information model also defines metadata about the data origin via the Subject Statement-Metadata. Typical examples of metadata are time stamps, data origin or data source.

Posture: Defined in [RFC5209] as "configuration and/or status of hardware or software on an endpoint as it pertains to an organization’s security policy."

This term is used within the scope of SACM to represent the configuration and state information that is collected from a target endpoint in the form of endpoint attributes (e.g. software/hardware inventory, configuration settings, dynamically assigned addresses). This information may constitute one or more posture attributes.

Posture Attributes: Defined in [RFC5209] as "attributes describing the configuration or status (posture) of a feature of the endpoint. A Posture Attribute represents a single property of an observed state. For example, a Posture Attribute might describe the version of the operating system installed on the system."

Within this document this term represents a specific assertion about endpoint configuration or state (e.g. configuration setting, installed software, hardware) represented via endpoint attributes. The phrase "features of the endpoint" highlighted above refers to installed software or software components.

Provider: A provider is a SACM role assigned to a SACM component that provides role-specific functions to provide information to other SACM components.

Repository: A repository is a controller that contains functions to consume, store and provide information of a particular kind.

Such information is typically data transported on the data plane, but potentially also data and metadata from the control and management plane. A single repository may provide the functions of more than one specific repository type (i.e. configuration baseline repository, assessment results repository, etc.)

SACM Broker Controller: A SACM Broker Controller is a controller that contains control plane functions to provide and/or connect services on behalf of other SACM components via interfaces on the control plane.
A broker may provide, for example, authorization services and find, upon request, SACM components providing requested services.

**SACM Component:** Is a component, as defined in [I-D.ietf-i2nsf-terminology], that is composed of SACM capabilities.

In the context of SACM, a set of SACM functions composes a SACM component. A SACM component conducts SACM tasks, acting on control plane, data plane and/or management plane via corresponding SACM interfaces. SACM defines a set of standard components (e.g. a collector, a broker, or a data store). A SACM component contains at least a basic set of control plane functions and can contain data plane and management plane functions. A SACM component residing on an endpoint assigns one or more SACM roles to the corresponding endpoint due to the SACM functions it is composed of. A SACM component "resides on" an endpoint and an endpoint "contains" a SACM component, correspondingly. For example, a SACM component that is composed solely of functions that provide information would only take on the role of a provider.

**SACM Component Discovery:** The task of discovering the capabilities provided by SACM components within a SACM domain.

This is likely to be performed via an appropriate set of control plane functions.

**SACM Component Label:** A specific endpoint label that is used to identify a SACM component.

In content-metadata, this label is called data origin.

**SACM Content:** The payload provided by SACM components to the SACM domain on the data plane.

SACM content includes the SACM data models.

**SACM Domain:** Endpoints that include a SACM component compose a SACM domain.

(To be revised, additional definition content TBD, possible dependencies to SACM architecture)

**SACM Function:** A behavioral aspect of a SACM component that provides external SACM Interfaces or internal interfaces to other SACM Functionse.
For example, a SACM Function with SACM Interfaces on the Control Plane can provide a brokering function to other SACM Components. Via Data Plane interfaces, a SACM Function can act as a provider and/or as a consumer of information. SACM Functions can be propagated as the Capabilities of a SACM Component and can be discovered by or negotiated with other SACM Components.

**SACM Interface:** An interface, as defined in [I-D.ietf-i2nsf-terminology], that provides SACM-specific operations.

[I-D.ietf-i2nsf-terminology] defines interface as a "set of operations one object knows it can invoke on, and expose to, another object," and further defines interface by stating that an interface "decouples the implementation of the operation from its specification. An interface is a subset of all operations that a given object implements. The same object may have multiple types of interfaces to serve different purposes."

In the context of SACM, SACM Functions provide SACM Interfaces on the management, control, or data plane. Operations a SACM Interface provides are based on corresponding data model defined by SACM. SACM Interfaces are used for communication between SACM components.

**SACM Proxy Controller:** A SACM Proxy Controller is a controller that provides data plane and control plane functions, information, or services on behalf of another component, which is not directly participating in the SACM architecture.

**SACM Role:** Is a role, as defined in [I-D.ietf-i2nsf-terminology], that requires the SACM Component assuming the role to bear a set of SACM functions or interfaces.

SACM Roles provide three important benefits. First, it enables different behavior to be supported by the same Component for different contexts. Second, it enables the behavior of a Component to be adjusted dynamically (i.e., at runtime, in response) to changes in context, by using one or more Roles to define the behavior desired for each context. Third, it decouples the Roles of a Component from the Applications that use that Component.

In the context of SACM, SACM roles are associated with SACM components and are defined by the set of functions and interfaces a SACM component includes. There are three SACM roles: provider, consumer, and controller. The roles associated with a SACM
component are determined by the purpose of the SACM functions and corresponding SACM interfaces the SACM component is composed of.

SACM Statement: Is an assertion that is made by a SACM Component.

Security Automation: The process of which security alerts can be automated through the use of different components to monitor, analyze and assess endpoints and network traffic for the purposes of detecting misconfigurations, misbehaviors or threats.

Security Automation is intended to identify target endpoints that cannot be trusted (see "trusted" in [RFC4949]). This goal is achieved by creating and processing evidence (assessment statements) that a target endpoint is not a trusted system [RFC4949].

Software Package: A generic software package (e.g. a text editor).

Software Component: A software package installed on an endpoint.

The software component may include a unique serial number (e.g. a text editor associated with a unique license key).

Software Instance: A running instance of a software component.

For example, on a multi-user system, one logged-in user has one instance of a text editor running and another logged-in user has another instance of the same text editor running, or on a single-user system, a user could have multiple independent instances of the same text editor running.

State: A volatile set of endpoint attributes of a (target) endpoint that is affected by a reboot-cycle.

Local state is created by the interaction of components with other components via the control plane, via processing data plane payload, or via the functional properties of local hardware and software components. Dynamic configuration (e.g. IP address distributed dynamically via an address distribution and management services, such as DHCP) is considered state that is the result of the interaction with another component (e.g. provided by a DHCP server with a specific configuration).

Examples: The static association of an IP address and a MAC address in a DHCP server configuration, a directory-path that identifies a log-file directory, a registry entry.
Statement: A statement is the root/top-level subject defined in the SACM information model.

A statement is used to bundle Content Elements into one subject and includes metadata about the data origin.

Subject: A semantic composite information element pertaining to a system entity that is a target endpoint.

Like Attributes, subjects have a name and are composed of attributes and/or other subjects. Every IE that is part of a subject can have a quantity associated with it (e.g. zero-one, none-unbounded). The content IE of a subject can be an unordered or an ordered list.

In contrast to the definitions of subject provided by [RFC4949], a subject in the scope of SACM is neither "a system entity that causes information to flow among objects or changes the system state" nor "a name of a system entity that is bound to the data items in a digital certificate".

In the context of SACM, a subject is a semantic composite of information elements about a system entity that is a target endpoint. Every acquirable subject—as defined in the scope of SACM—about a target endpoint represents and therefore identifies every subject—as defined by [RFC4949]—that is a component of that target endpoint. The semantic difference between both definitions can be subtle in practice and is in consequence important to highlight.

Supplicant: A component seeking to be authenticated via the control plane for the purpose of participating in a SACM domain.

System Resource: Defined in [RFC4949] as "data contained in an information system; or a service provided by a system; or a system capacity, such as processing power or communication bandwidth; or an item of system equipment (i.e., hardware, firmware, software, or documentation); or a facility that houses system operations and equipment."

Target Endpoint: Is an endpoint that is under assessment at some point in, or region of, time.

Every endpoint that is not specifically designated as an excluded endpoint is a target endpoint. A target endpoint is not part of a SACM domain unless it contains a SACM component (e.g. a SACM component that publishes collection results coming from an internal collector).
A target endpoint is similar to a device that is a Target of Evaluation (TOE) as defined in Common Criteria and as referenced by {RFC4949}.  

Target Endpoint Address: An address that is layer specific and which follows layer specific address schemes.  

Each interface of a specific layer can be associated with one or more addresses appropriate for that layer. There is no guarantee that an address is globally unique. In general, there is a scope to an address in which it is intended to be unique.  

Examples include: physical Ethernet port with a MAC address, layer 2 VLAN interface with a MAC address, layer 3 interface with multiple IPv6 addresses, layer 3 tunnel ingress or egress with an IPv4 address. 

Target Endpoint Characterization: The description of the distinctive nature of a target endpoint, that is based on its characteristics.  

Target Endpoint Characterization Record: A set of endpoint attributes about a target endpoint that was encountered in a SACM domain, which are associated with that target endpoint as a result of a Target Endpoint Characterization Task.  

A characterization record is intended to be a representation of an endpoint. It cannot be assured that a record distinctly represents a single target endpoint unless a set of one or more endpoint attributes that compose a unique set of identifying endpoint attributes are included in the record. Otherwise, the set of identifying attributes included in a record can match more than one target endpoints, which are - in consequence - indistinguishable to a SACM domain until more qualifying endpoint attributes can be acquired and added to the record. A characterization record is maintained over time in order to assert that acquired endpoint attributes are either about an endpoint that was encountered before or an endpoint that has not been encountered before in a SACM domain. A characterization record can include, for example, acquired configuration, state or observed behavior of a specific target endpoint. Multiple and even conflicting instances of this information can be included in a characterization record by using timestamps and/or data origins to differentiate them. The endpoint attributes included in a characterization record can be used to re-identify a distinct target endpoint over time. Classes or profiles can be associated with a characterization record via the Classification Task in order to guide collection, evaluation or remediation tasks.
Target Endpoint Characterization Task: An ongoing task of continuously adding acquired endpoint attributes to a corresponding record. The TE characterization task manages the representation of encountered target endpoints in the SACM domain in the form of characterization records. For example, the output of a target endpoint discovery task or a collection task can be processed by the characterization task and added to the record. The TE characterization Task also manages these representations of target endpoints encountered in the SACM domain by splitting or merging the corresponding records as new or more refined endpoint attributes become available.

Target Endpoint Classification Task: The task of associating a class from an extensible list of classes with an endpoint characterization record. TE classes function as imperative and declarative guidance for collection, evaluation, remediation and security posture assessment in general.

Target Endpoint Discovery Task: The ongoing task of detecting previously unknown interaction of a potential target endpoint in the SACM domain. TE Discovery is not directly targeted at a specific target endpoint and therefore an un-targeted task. SACM Components conducting the discovery task as a part of their function are typically distributed and located, for example, on infrastructure components or collect from those remotely via appropriate interfaces. Examples of infrastructure components that are of interest to the discovery task include routers, switches, VM hosting or VM managing components, AAA servers, or servers handling dynamic address distribution.

Target Endpoint Identifier: The target endpoint discovery task and the collection tasks can result in a set of identifying endpoint attributes added to a corresponding Characterization Record. This subset of the endpoint attributes included in the record is used as a target endpoint identifier, by which a specific target endpoint can be referenced. Depending on the available identifying attributes, this reference can be ambiguous and is a "best-effort" mechanism. Every distinct set of identifying endpoint attributes can be associated with a target endpoint label that is unique in a SACM domain.

Target Endpoint Label: An endpoint label that identifies a specific target endpoint.

Target Endpoint Profile: A bundle of expected or desired component composition, configurations and states that is associated with a target endpoint.
The corresponding task by which the association with a target endpoint takes places is the endpoint classification task. The task by which an endpoint profile is created is the endpoint characterization task. A type or class of target endpoints can be defined via a target endpoint profile. Examples include: printers, smartphones, or an office PC.

In respect to [RFC4949], a target endpoint profile is a protection profile as defined by Common Criteria (analogous to the target endpoint being the target of evaluation).

**SACM Task:** Is a task conducted within the scope of a SACM domain by one or more SACM functions that achieves a SACM-defined outcome.

A SACM task can be triggered by other operations or functions (e.g. a query from another SACM component or an unsolicited push on the data plane due to an ongoing subscription). A task is part of a SACM process chain. A task starts at a given point in time and ends in a deterministic state. With the exception of a collection task, a SACM task consumes SACM statements provided by other SACM components. The output of a task is a result that can be provided (e.g. published) on the data plane.

The following tasks are defined by SACM:

- Target Endpoint Discovery
- Target Endpoint Characterization
- Target Endpoint Classification
- Collection
- Evaluation [TBD]
- Information Sharing [TBD]
- SACM Component Discovery
- SACM Component Authentication [TBD]
- SACM Component Authorization [TBD]
- SACM Component Registration [TBD]

**Timestamps:** Defined in [RFC4949] as "with respect to a data object, a label or marking in which is recorded the time (time of day or
other instant of elapsed time) at which the label or marking was affixed to the data object”.

A timestamp always requires context, i.e. additional information elements that are associated with it. Therefore, all timestamps wrt information elements are always metadata. Timestamps in SACM Content Elements may be generated outside a SACM Domain and may be encoded in an unknown representation. Inside a SACM domain the representation of timestamps is well-defined and unambiguous.

Virtual Endpoint: An endpoint composed entirely of logical system components (see [RFC4949]).

The most common example is a virtual machine/host running on a target endpoint. Effectively, target endpoints can be nested and at the time of this writing the most common example of target endpoint characteristics about virtual components is the EntLogicalEntry in [RFC6933].

Vulnerability Assessment: An assessment specifically tailored to determining whether a set of endpoints is vulnerable according to the information contained in the vulnerability description information.

Vulnerability Description Information: Information pertaining to the existence of a flaw or flaws in software, hardware, and/or firmware, which could potentially have an adverse impact on enterprise IT functionality and/or security.

Vulnerability description information should contain enough information to support vulnerability detection.

Vulnerability Detection Data: A type of imperative guidance extracted or derived from vulnerability description information that describes the specific mechanisms of vulnerability detection that is used by an enterprise’s vulnerability management capabilities to determine if a vulnerability is present on an endpoint.

Vulnerability Management Capabilities: An IT management capability tailored toward managing endpoint vulnerabilities and associated metadata on an ongoing basis by ingesting vulnerability description information and vulnerability detection data, and performing vulnerability assessments.

Vulnerability assessment capabilities: An assessment capability that is tailored toward determining whether a set of endpoints is vulnerable according to vulnerability description information.
Workflow: A workflow is a modular composition of tasks that can contain loops, conditionals, multiple starting points and multiple endpoints.

The most prominent workflow in SACM is the assessment workflow.

3. IANA Considerations

This memo includes no request to IANA.

4. Security Considerations

This memo documents terminology for security automation. While it is about security, it does not affect security.

5. Acknowledgements

6. Change Log

Changes from version 00 to version 01:

- Added simple list of terms extracted from UC draft -05. It is expected that comments will be received on this list of terms as to whether they should be kept in this document. Those that are kept will be appropriately defined or cited.

Changes from version 01 to version 02:

- Added Vulnerability, Vulnerability Management, xposure, Misconfiguration, and Software flaw.

Changes from version 02 to version 03:

- Removed Section 2.1. Cleaned up some editing nits; broke terms into 2 sections (predefined and newly defined terms). Added some of the relevant terms per the proposed list discussed in the IETF 89 meeting.

Changes from version 03 to version 04:

- TODO

Changes from version 04 to version 05:

- TODO

Changes from version 05 to version 06:
Updated author information.

Combined "Pre-defined Terms" with "New Terms and Definitions".

Removed "Requirements language".

Removed unused reference to use case draft; resulted in removal of normative references.

Removed introductory text from Section 1 indicating that this document is intended to be temporary.

Added placeholders for missing change log entries.

Changes from version 06 to version 07:

- Added Contributors section.
- Updated author list.
- Changed title from "Terminology for Security Assessment" to "Secure Automation and Continuous Monitoring (SACM) Terminology".
- Changed abbrev from "SACM-Terms" to "SACM Terminology".
- Added appendix The Attic to stash terms for future updates.
- Added Authentication, Authorization, Data Confidentiality, Data Integrity, Data Origin, Data Provenance, SACM Component, SACM Component Discovery, Target Endpoint Discovery.
- Major updates to Building Block, Function, SACM Role, Target Endpoint.
- Relabeled Role to SACM Role, Endpoint Target to Target Endpoint, Endpoint Discovery to Endpoint Identification.
- Moved Asset Targeting, Client, Endpoint Identification to The Attic.
- Endpoint Attributes added as a TODO.
- Changed the structure of the Change Log.

Changes from version 07 to version 08:
o Added Assertion, Collection Result, Collector, Excluded Endpoint, Internal Collector, Network Address, Network Interface, SACM Domain, Statement, Target Endpoint Identifier, Target Endpoint Label, Timestamp.

o Major updates to Attributes, Broker, Collection Task, Consumer, Controller, Control Plane, Endpoint Attributes, Expected Endpoint State, SACM Function, Provider, Proxy, Repository, SACM Role, Target Endpoint.

o Minor updates to Asset, Building Block, Data Origin, Data Source, Data Provenance, Endpoint, Management Plane, Posture, Posture Attribute, SACM Component, SACM Component Discovery, Target Endpoint Discovery.

o Relabeled Function to SACM Function.

Changes from version 08 to version 09:

o Updated author list.

o Added Data Plane, Endpoint Characterization, Endpoint Classification, Guidance, Interaction Model, Software Component, Software Instance, Software Package, Statement, Target Endpoint Profile, SACM Task.

o Removed Building Block.

o Major updates to Control Plane, Endpoint Attribute, Expected Endpoint State, Information Model, Management Plane.

o Minor updates to Attribute, Capabilities, SACM Function, SACM Component, Collection Task.

o Moved Asset Characterization to The Attic.

Changes from version 09 to version 10:

o Added Configuration Drift, Data in Motion, Data at Rest, Endpoint Management Capability, Hardware Component, Hardware Inventory, Hardware Type, SACM Interface, Target Endpoint Characterization Record, Target Endpoint Characterization Task, Target Endpoint Classification Task, Target Endpoint Discovery Task, Vulnerability Description Information, Vulnerability Detection Data, Vulnerability Management Capability, Vulnerability Assessment

o Added references to i2nsf definitions in Capability, SACM Component, SACM Interface, SACM Role.
o Added i2nsf Terminology I-D Reference.

o Major Updates to Endpoint, SACM Task, Target Endpoint Identifier.

o Minor Updates to Guidance, SACM Component Discovery, Target Endpoint Label, Target Endpoint Profile.

o Relabeled SACM Task

o Removed Target Endpoint Discovery

Changes from version 10 to version 11:

o Added Content Element, Content Metadata, Endpoint Label, Information Element, Metadata, SACM Component Label, Workflow.

o Major Updates to Assessment, Capability, Collector, Endpoint Management Capabilities, Guidance, Vulnerability Assessment Capabilities, Vulnerability Detection Data, Vulnerability Assessment Capabilities.

o Minor updates to Collection Result, Control Plane, Data in Motion, Data at Rest, Data Origin, Network Interface, Statement, Target Endpoint Label.

o Relabeled Endpoint Management Capability, Vulnerability Management Capability, Vulnerability Assessment.

Changes from version 11 to version 12:

o Added Configuration, Endpoint Characteristic, Event, SACM Content, State, Subject.

o Major Updates to Assertion, Data in Motion, Data Provenance, Data Source, Interaction Model.

o Minor Updates to Attribute, Control Plane, Data Origin, Data Provenance, Expected Endpoint State, Guidance, Target Endpoint Classification Task, Vulnerability Detection Data.

Changes from version 12 to version 13:

o Added Virtual Component.

o Major Updates to Capability, Collection Task, Hardware Component, Hardware Type, Security Automation, Subject, Target Endpoint, Target Endpoint Profile.
o Minor Updates to Assertion, Data Plane, Endpoint Characteristics.

Changes from version 13 to version 14:

o Handled a plethora of issues listed in GitHub.

o Pruned some commonly understood terms.

o Narrowing term labels per their definitions.

o In some cases, excised expositional text.

o Where expositional text was left intact, it has been separated from the actual definition of a term.

7. Contributors
8. References

Birkholz, et al. Expires December 15, 2018
8.1. Normative References


8.2. Informative References


Appendix A. The Attic

The following terms are stashed for now and will be updated later:

Asset Characterization: Asset characterization is the process of defining attributes that describe properties of an identified asset.

Asset Targeting: Asset targeting is the use of asset identification and categorization information to drive human-directed, automated decision making for data collection and analysis in support of endpoint posture assessment.

Client: An architectural component receiving services from another architectural component.

Endpoint Identification (TBD per list; was "Endpoint Discovery"): The process by which an endpoint can be identified.

Authors’ Addresses

Henk Birkholz
Fraunhofer SIT
Rheinstrasse 75
Darmstadt 64295
Germany

Email: henk.birkholz@sit.fraunhofer.de

Jarrett Lu
Oracle Corporation
4180 Network Circle
Santa Clara, CA 95054
USA

Email: jarrett.lu@oracle.com

John Strassner
Huawei Technologies
2330 Central Expressway
Santa Clara, CA 95138
USA

Email: john.sc.strassner@huawei.com
Abstract

This document extends the Resource-Oriented Lightweight Information Exchange (ROLIE) core by defining a new information-type to ROLIE’s atomcategory pertaining to security configuration checklists. Additional supporting requirements are also defined which describe the use of specific formats and link relations pertaining to the new information-type.

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 16, 2019.

Copyright Notice

Copyright (c) 2018 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust’s Legal Provisions Relating to IETF Documents (https://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must
This document defines an extension to the Resource-Oriented Lightweight Information Exchange (ROLIE) [RFC8322] protocol [RFC8322] to support the publication of configuration checklist information.

Many enterprises operate according to guidance provided to them by a control framework ([CIS_Critical_Controls], [PCI_DSS], [NIST_800-53] etc.), which often prescribe that an enterprise define a standard, secure configuration for each technology they operate. Such standard secure configurations are often referred to as configuration checklists. These configuration checklists contain a set of configuration recommendations for a given endpoint. A configuration recommendation prescribes expected values pertaining to one or more discrete endpoint attributes.

2. Terminology

Configuration Checklist A configuration checklist is an organized collection of rules about a particular kind of system or platform.

Configuration Item Generally synonymous with endpoint attribute.
A configuration recommendation is an expression of the desired posture of one or more configuration items. A configuration recommendation generally includes the description of the recommendation, a rationale statement, and the expected state of collected posture information.

TODO: Others??  TBD

TODO: There needs to be a "normative" reference to the SCAP 1.2/3 specifications and schema definitions

3. The 'configuration-checklist' information type

This document defines and registers a new information-type: "configuration-checklist".

The "configuration-checklist" information type represents a body of information describing a set of configuration recommendations. A configuration recommendation is, minimally, a configurable item paired with a recommended value or range of value. Depending on the source, a configuration recommendation may carry with it additional information (i.e. description, references, rationale, etc.). Provided below is a non-exhaustive list of information that may be considered as components of a configuration checklist.

- A "Data Stream"
- A "Benchmark"
- A "Profile"
- A "Value"
- A "Rule" or "Group" of Rules
  - Description
  - Rationale
  - Remediation Instructions
  - Information, described in the dialect of a supported "check system", indicating the method(s) used to audit the checklist configuration item.
- Applicable Platform Information
- Information regarding a set of patches to be evaluated
4. Data format requirements

This section defines usage guidance and additional requirements related to data formats above and beyond those specified in [RFC8322]. The following formats are expected to be commonly used to express software descriptor information. For this reason, this document specifies additional requirements to ensure interoperability.

TODO, integrate this information:

- scap-1.2
- PDF
- xccdf-1.2-collection
- oval
- cvrf
- cve (should we reuse the enumref?)

Look at the "enumref" and see if we can copy/paste configuration checklist-specific information in a similar manner? Can we then include that enum reference in the ROLIE extension document or should we create a new "enumref" document separately?

- vulnerability

4.1. Data Format 1

4.1.1. Description

This is data section 1 TODO

4.1.2. Requirements

This is requirement 1 TODO
5. rolie:property Extensions

This document provides new registrations for valid rolie:property names. These properties provide optional exposure point for valuable information in the linked content document. Exposing this information in a rolie:property element means that clients do not need to download the linked document to determine if it contains information they are interested in.

A breadth of metadata may be included with a configuration checklist as identifying information. A publishing organization may wish to recognize or attribute checklist authors or contributors, or maintain a revision/version history over time. Other metadata that may be included could indicate the various categories of products to which the checklist applies, such as Operating System, Network Device, or Application Server.

The following list describes various ‘rolie:property’ constructs.

- contributor (0..n)
  * An unbounded number of "rolie:property" elements with a "name" attribute of "contributor" may be included to indicate those individuals noted as recognized contributors to the configuration checklist and/or the recommendations contained within.

- checklist version: The "value" of the "checklist version" property indicates the version number of the configuration checklist, such as "3.1.1"

- title: The "value" of the "title" property indicates the document title of the configuration checklist, such as "CIS Benchmark for Microsoft Windows Server 2012 R2"

- overview

6. Use of the atom:link element

The following link relations are defined in the following table. These relations are not registered in the Link Relation IANA table due to their niche usage. These link relations are valid for any link element in a checklist Entry.
<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ancestor</td>
<td>Links to a configuration checklist superseded by that described in this entry</td>
</tr>
<tr>
<td>target-platform</td>
<td>Links to a software descriptor resource defining the software subject to this configuration checklist entry</td>
</tr>
<tr>
<td>version</td>
<td>Links to a text resource indicating the version of the configuration checklist</td>
</tr>
</tbody>
</table>

7. Use of atom:category


When the name attribute of a category element is this names, the value attribute SHOULD be one of the valid product categories from the NIST NCP Product Category List, such as:

- Antivirus Software
- Application Server
- Auditing
- Authentication
- Automation/Productivity Application Suite
- Client and Server Encryption
- Configuration Management Software
- Database Management System
- Desktop Application
- Desktop Client
- DHCP Server
- Directory Service
- DNS Server
- Email Server
- Encryption Software
- Enterprise Application
- File Encryption
- Firewall
- Firmware
- Handheld Device
- Identity Management
- Intrusion Detection System
- KVM
- Mail Server
- Malware
- Mobile Solution
- Monitoring
- Multi-Functional Peripheral
- Network Router
- Network Switch
- Office Suite
- Operating System
- Peripheral Device
- Security Server
- Server
- Virtual Machine
- Virtualization Software
8. IANA Considerations

Per this document, IANA has added an entry to the "ROLIE Security Resource Information Type Sub-Registry" registry located at https://www.iana.org/assignments/rolie/category/information-type [1].

name: configuration-checklist
index: TBD
reference: This document, Section TODO
TODO add Propertyies and Categories

9. Security Considerations

Any user of this extension should be familiar with the security considerations of ROLIE [RFC8322].

10. Privacy Considerations

Any user of this extension should be familiar with the privacy considerations of ROLIE [RFC8322].

11. References

11.1. Normative References


11.2. Informative References

11.3. URIs

[1] https://www.iana.org/assignments/rolie/category/information-type

Authors' Addresses

Bill Munyan
Center for Internet Security
31 Tech Valley Drive
East Greenbush, NY 12061
USA
Email: bill.munyan.ietf@gmail.com

Adam Montville
Center for Internet Security
31 Tech Valley Drive
East Greenbush, NY 12061
USA
Email: adam.w.montville@gmail.com

Stephen A. Banghart
National Institute of Standards and Technology
100 Bureau Drive
Gaithersburg, Maryland
USA
Phone: (301) 975-4288
Email: sab3@nist.gov