

SACM Working Group
Internet-Draft
Intended status: Standards Track
Expires: 10 January 2022

A. Montville
B. Munyan
CIS
9 July 2021

Security Automation and Continuous Monitoring (SACM) Architecture
draft-ietf-sacm-arch-13

Abstract

This document defines an architecture enabling a cooperative Security Automation and Continuous Monitoring (SACM) ecosystem. This work is predicated upon information gleaned from SACM Use Cases and Requirements ([RFC7632] and [RFC8248] respectively), and terminology as found in [I-D.ietf-sacm-terminology].

WORKING GROUP: The source for this draft is maintained in GitHub. Suggested changes should be submitted as pull requests at <https://github.com/sacmwg/ietf-mandm-sacm-arch/>. Instructions are on that page as well.

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 10 January 2022.

Copyright Notice

Copyright (c) 2021 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	4
1.1. Requirements notation	4
2. Terms and Definitions	4
3. Architectural Overview	8
3.1. Producer	9
3.2. Consumer	9
3.3. Integration Service	9
3.4. Payload/Message	10
3.5. Payload Categorization	10
3.5.1. Topic-centric	10
3.5.2. Payload-centric	11
3.6. Capabilities	11
3.7. Interaction Categories	12
3.7.1. Broadcast	12
3.7.2. Directed	12
4. SACM Role-based Architecture	13
4.1. Architectural Roles/Components	14
4.1.1. Manager	14
4.1.2. Orchestrator(s)	14
4.1.3. Repositories	15
4.1.4. Integration Service	15
4.2. Downstream Uses	16
4.2.1. Reporting	16
4.2.2. Analytics	16
4.3. Sub-Architectures	16
4.3.1. Collection Sub-Architecture	16
4.3.2. Evaluation Sub-Architecture	19
5. Ecosystem Interactions	21
5.1. Manager	21
5.2. Component Registration	22
5.3. Administrative Interface	23
5.3.1. Capability Advertisement Handshake	23
5.3.2. Health Check	23
5.3.3. Heartbeat	23
5.3.4. Capability-specific Requests	24
5.4. Status Notifications	24
5.5. Component Interactions	24
5.5.1. Initiate Ad-Hoc Collection	24
5.5.2. Coordinate Periodic Collection	24
5.5.3. Coordinate Observational/Event-based Collection	25
5.5.4. Persist Collected Posture Attributes	26

5.5.5.	Initiate Ad-Hoc Evaluation	26
5.5.6.	Coordinate Periodic Evaluation	26
5.5.7.	Coordinate Change-based Evaluation	27
5.5.8.	Queries	27
6.	Operations	27
6.1.	Component Registration	27
6.1.1.	Request Payload	28
6.1.2.	Request Processing	28
6.1.3.	Response Payload	29
6.1.4.	Response Processing	29
6.2.	Administrative Interface	29
6.2.1.	Capability Advertisement Handshake	29
6.2.2.	Health Check	31
6.2.3.	Heartbeat	32
6.3.	Status Notification	33
6.3.1.	Request Payload	34
6.3.2.	Request Processing	34
6.3.3.	Response Payload	34
6.3.4.	Response Processing	34
6.4.	Initiate Ad-Hoc Collection	34
6.4.1.	SACM Producer to Orchestrator	36
6.4.2.	Orchestrator to Posture Collection Service	36
6.4.3.	Posture Collection Service to Posture Attribute Repository	38
6.5.	Initiate Ad-Hoc Evaluation	39
7.	Privacy Considerations	39
8.	Security Considerations	39
9.	IANA Considerations	39
9.1.	Component Types	39
9.2.	Component Capabilities	40
9.2.1.	Heartbeat	40
9.2.2.	Status Notification (Publish)	40
9.2.3.	Status Notification (Subscribe)	40
10.	References	40
10.1.	Normative References	40
10.2.	Informative References	41
Appendix A.	Security Domain Workflows	43
A.1.	IT Asset Management	43
A.1.1.	Components, Capabilities and Workflow(s)	44
A.2.	Vulnerability Management	44
A.2.1.	Components, Capabilities and Workflow(s)	45
A.3.	Configuration Management	46
A.3.1.	Components, Capabilities and Workflow(s)	47
Authors' Addresses	49

1. Introduction

The purpose of this draft is to define an architectural approach for a SACM Domain, based on the spirit of use cases found in [RFC7632] and requirements found in [RFC8248]. This approach gains the most advantage by supporting a variety of collection systems, and intends to enable a cooperative ecosystem of tools from disparate sources with minimal operator configuration.

1.1. 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. Terms and Definitions

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.

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

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

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 a 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.

Consumer: A SACM Role that requires a SACM Component to include SACM Functions enabling it to receive information from other SACM Components.

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.

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.)

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].

SIEM: TBD

SOAR: TBD

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).

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].

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.

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. Architectural Overview

The generic approach proposed herein recognizes the need to obtain information from existing and future state collection systems, and makes every attempt to respect [RFC7632] and [RFC8248]. At the foundation of any architecture are entities, or components, that need to communicate. They communicate by sharing information, where, in a given flow, one or more components are consumers of information and one or more components are providers of information. Different roles within a cooperative ecosystem may act as both Producers and Consumers of SACM-relevant information.

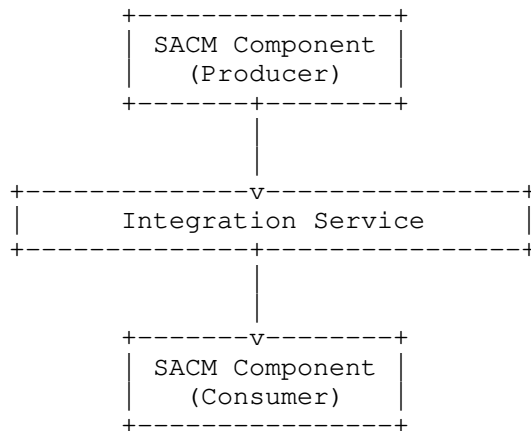


Figure 1: Basic Architectural Structure

3.1. Producer

A Producer can be described as an abstraction that refers to an entity capable of sending SACM-relevant information to one or many Consumers. In general, information (a "payload") is produced to a particular topic, subscribed to by one or more Consumers. Producers need not be concerned about any specifics of the payload it is providing to a given topic. A Producer may, for example, publish posture collection instructions to collector topics.

3.2. Consumer

A Consumer can be described as an abstraction that refers to an entity capable of receiving SACM-relevant information from one or many Producers. A Consumer acts as a subscriber to a given topic (or set of topics), enabling it to receive event notifications when a Producer provides a payload to that topic or topics. Consumers receive payloads and act upon them according to their capabilities. A Consumer may, for example, subscribe to a posture collection topic to receive and act upon, collection instructions.

3.3. Integration Service

The Integration Service acts as the broker between Producers and Consumers; acting as the destination for Producers to publish payloads, and as the source for Consumers subscribing to those payloads.

SACM Components are intended to interact with other SACM Components. These interactions can be thought of, at the architectural level, as the combination of interfaces with their supported operations. Each interaction will convey a classified payload of information. This classification of payload information allows Consumers to subscribe to only the classifications to which they are capable of handling. The payload information should contain subdomain-specific characteristics and/or instructions.

3.4. Payload/Message

The payload (sometimes referred to as a "message" or "message payload") is the unit of data involved in any given interaction between two SACM components. The payload MAY be used to convey the semantic meaning of the operation to be performed. Protocols such as [RFC6120] achieves this meaning through XML namespace identification within a "<message/>" or "<iq/>" stanza. Topic-centric protocols such as [MQTT] convey the meaning of payloads through topic naming techniques. Both methods require connected components to verify message payloads according to their respective capabilities.

With respect to the Integration Service, the payload is simply an array of bytes, so the data contained within it is not required to convey a specific format or meaning to the Integration Service. The serialization of the payload combined with the payload categorization provides meaning within the SACM context.

3.5. Payload Categorization

Within the SACM ecosystem, categorization of payloads and their transport provide the context through which various capabilities are achieved. Two types of payload categorization can be described.

3.5.1. Topic-centric

Topic-centric payload categorization allows for a broad spectrum of payloads by characterizing those payloads through the Integration Service topic. In this categorization, the topic name becomes a label attached to the payload to which the Integration Service matches against known subscriptions. The topic becomes the operational context for the payload. Topic-centric categorization allows for any payload to be sent to any topic, but requires that SACM consumers parse the payloads to determine whether or not they have the capability to act on those payloads.

When interacting using a topic-centric payload categorization, topic naming conventions SHOULD provide an adequate amount of information to be deterministic regarding the purpose of the interaction. For

example, a topic named `"/notification/collection/oval"` would indicate that (a) the topic is a broadcast/notification (publish/subscribe) topic, (b) subscribers to this topic are performing a "collection" action, and (c) the payloads published to the topic are represented using the OVAL serialization format.

3.5.2. Payload-centric

Payload-centric categorization encapsulates the intent of an interaction within the message payload itself, using an identifying token, tag, or namespace identifier. This method allows for the limitation of message types, and therefore increases the extensibility of message payloads.

Payload-centric categorization allows for modularization and specification of extensions, and for plugin-based support of capabilities based the categorization. XMPP is an example of utilization of payload-centric categorization, allowing only three distinct "stanzas" ("`<message/>`", "`<presence/>`", and "`<iq/>`"), using payloads defined by the various extension protocols maintained by the XMPP standards foundation.

3.6. Capabilities

SACM components interact with each other based on their capacity to perform specific actions. In advertising its capabilities, a SACM component indicates its competence to understand message payloads, perform any payload translation or normalization, and act upon that message. For example, an Orchestration component receives a message to initiate posture attribute collection. The Orchestrator may then normalize those instructions to a particular collection system's serialization. The normalized instructions are then published to the Integration Service, notifying the appropriate subscribers.

Capabilities are described using Uniform Resource Names (URNs), which will be maintained and enhanced via IANA tables (IANA Considerations). Using topic-centric categorization of message payloads, capability URNs SHOULD be associated with Integration Service topics to which publishers, subscribers, and service handlers, will interact. Topic naming conventions are considered implementation details and are not considered for standardization. Given a payload-centric categorization of message payloads, capability URNs SHOULD be used as the identifying token, tag, or namespace in order to distinguish specific payloads.

3.7. Interaction Categories

Two categories of interactions SHOULD be supported by the Integration Service: broadcast and directed. Broadcast interactions are asynchronous by default, and directed interactions may be invoked either synchronously or asynchronously.

3.7.1. Broadcast

A broadcast interaction, commonly referred to as publish/subscribe, allows for a wider distribution of a message payload. When a payload is published to the Integration Service, all subscribers to that payload are alerted and may consume the message payload. This category of interaction can also be described as a "unicast" interaction when only a single subscriber exists. An example of a broadcast interaction could be to publish Linux OVAL objects to a posture collection topic. Subscribing consumers receive the notification, and proceed to collect endpoint configuration posture based on the supplied message payload.

3.7.2. Directed

The intent of a directed interaction is to enable point-to-point communications between a producer and consumer, through the standard interfaces provided by the Integration Service. The provider component indicates which consumer is intended to receive the payload, and the Integration Service routes the payload directly to that consumer. Two "styles" of directed interaction exist, differing only by the response from the consumer.

3.7.2.1. Synchronous

Synchronous, request/response style interaction requires that the requesting component block and wait for the receiving component to respond, or to time out when that response is delayed past a given time threshold. A synchronous interaction example may be querying a CMDB for posture attribute information in order to perform an evaluation.

3.7.2.2. Asynchronous

An asynchronous interaction involves the payload producer directing the message to a consumer, but not blocking or waiting for an immediate response. This style of interaction allows the producer to continue on to other activities without the need to wait for responses. This style is particularly useful when the interaction payload invokes a potentially long-running task, such as data collection, report generation, or policy evaluation. The receiving

component may reply later via callbacks or further interactions, but it is not mandatory.

4. SACM Role-based Architecture

Within the cooperative SACM ecosystem, a number of roles act in coordination to provide relevant policy/guidance, perform data collection, storage, evaluation, and support downstream analytics and reporting.

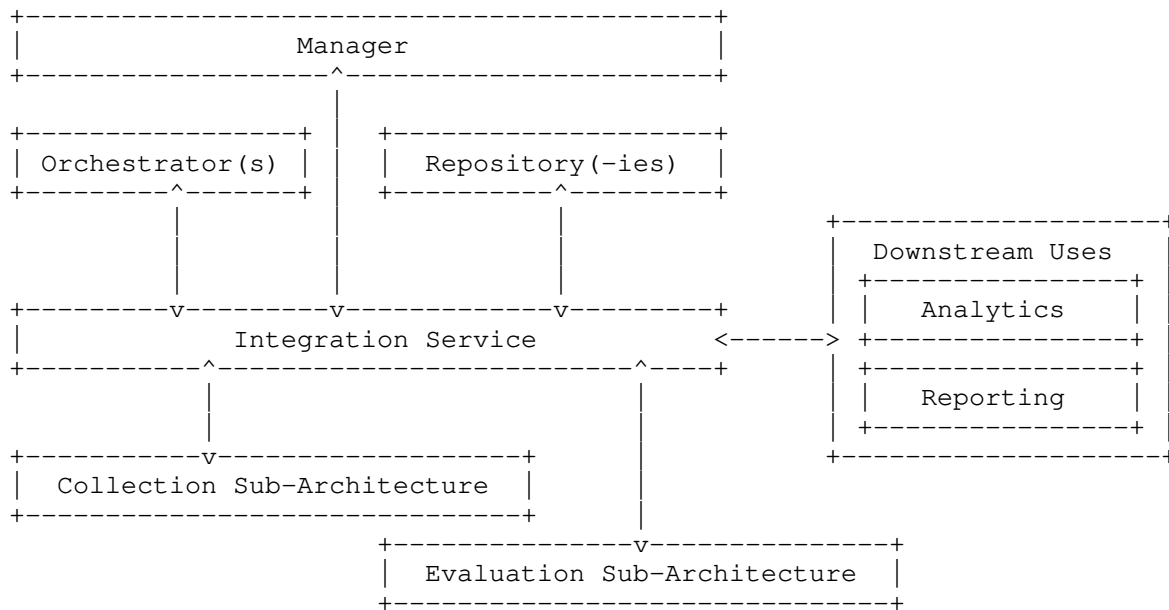


Figure 2: Notional Role-based Architecture

As shown in Figure 2, the SACM role-based architecture consists of some basic SACM Components communicating using an integration service. The integration service is expected to maximally align with the requirements described in [RFC8248], which means that the integration service will support brokered (i.e. point-to-point) and proxied data exchange.

4.1. Architectural Roles/Components

This document suggests a variety of players in a cooperative ecosystem; known as SACM Components. SACM Components may be composed of other SACM Components, and each SACM Component plays one, or more, of several roles relevant to the ecosystem. Roles may act as providers of information, consumers of information, or both provider and consumer. Figure 2 depicts a number of SACM components which are architecturally significant and therefore warrant discussion and clarification. Each role depicted in Figure 2 represents the interface to the component(s) fulfilling that role, not necessarily any specific implementation. For example, the "Repository" figure represents the interface to persistent storage, and not any particular persistent storage mechanism.

4.1.1. Manager

The Manager acts as the control plane for the SACM ecosystem; a sort of high level component capable of coordinating the actions, notifications, and events between components. The manager controls the administrative interfaces with the various components of the ecosystem, acting as the central point to which all other components will register and advertise their capabilities. It is the responsibility of the manager to control a component's access to the ecosystem, maintain an inventory of components attached to the ecosystem, and to initiate the various workflows involved in the collection and/or evaluation of posture attributes.

The manager should maintain the master set of capabilities that can be supported within the ecosystem. These are the various collection, evaluation, and persistence capabilities with which components may register. The manager MAY be responsible for assigning topics for each of the capabilities that are supported, as registering components subsequently subscribe to, or configure service handlers for, those topics.

The manager may act as the user interface to the ecosystem, providing user dashboards, inventories, component management, or operational controls within the boundary of responsibility.

4.1.2. Orchestrator(s)

Orchestration components provide the manager with resources for delegating work across the SACM ecosystem. Orchestrators are responsible for receiving messages from the manager, e.g. posture attribute collection instructions, and routing those messages to the appropriate "actions". For example, an orchestrator may support the capability of translating posture collection instructions using the

Open Vulnerability and Assessment Language (OVAL) and providing those instructions to OVAL collectors. An orchestrator may support the capability of initiating policy evaluation. Where the Manager is configured to ask a particular set of questions, those questions are delegated to Orchestrators, who are then capable of asking those questions using specific dialects.

4.1.3. Repositories

Figure 2 only includes a single reference to "Repository(-ies)", but in practice, a number of separate data repositories may exist, including posture attribute repositories, policy repositories, local vulnerability definition data repositories, and state assessment results repositories. The diagrammed notion of a repository within the SACM context represents an interface in which payloads are provided (based on the capabilities of the producer), normalized, and persisted.

These data repositories may exist separately or together in a single representation, and the design of these repositories may be as distinct as their intended purpose, such as the use of relational database management systems (RDBMS), filesystem-based storage, or graph/map implementations. Each implementation of a SACM repository should focus on the relationships between data elements and implement the SACM information and data model(s).

4.1.4. Integration Service

If each SACM component represents a set of capabilities, then the Integration Service represents the "fabric" by which SACM components are woven together. The Integration Service acts as a message broker, combining a set of common message categories and infrastructure to allow SACM components to communicate using a shared set of interfaces. The Integration Service's brokering capabilities enable the exchange of various information payloads, orchestration of component capabilities, message routing and reliable delivery. The Integration Service minimizes the dependencies from one system to another through the loose coupling of applications through messaging. SACM components will "attach" to the Integration Service either through native support for the integration implementation, or through the use of "adapters" which provide a proxied attachment.

The Integration Service should provide mechanisms for both synchronous and asynchronous request/response-style messaging, and a publish/subscribe mechanism to implement an event-based architecture. It is the responsibility of the Integration Service to coordinate and manage the sending and receiving of messages. The Integration Service should allow components to directly connect and produce or

consume messages, or connect via message translators which can act as a proxy, transforming messages from a component format to one implementing a SACM data model.

The Integration Service MUST provide routing capabilities for payloads between producers and consumers. The Integration Service MAY provide further capabilities within the payload delivery pipeline. Examples of these capabilities include, but are not limited to, intermediate processing, message transformation, type conversion, validation, or other enterprise integration patterns.

4.2. Downstream Uses

As depicted by Figure 2, a number of downstream uses exist in the cooperative ecosystem. Each notional SACM component represents distinct sub-architectures which will exchange information via the integration services, using interactions described in this draft.

4.2.1. Reporting

The Reporting component represents capabilities outside of the SACM architecture scope dealing with the query and retrieval of collected posture attribute information, evaluation results, etc. in various display formats that are useful to a wide range of stakeholders.

4.2.2. Analytics

The Analytics component represents capabilities outside of the SACM architecture scope dealing with the discovery, interpretation, and communication of any meaningful patterns of data in order to inform effective decision making within the organization.

4.3. Sub-Architectures

Figure 2 shows two components representing sub-architectural roles involved in a cooperative ecosystem of SACM components for the purpose of posture assessment: Collection and Evaluation.

4.3.1. Collection Sub-Architecture

The Collection sub-architecture is, in a SACM context, the mechanism by which posture attributes are collected from applicable endpoints and persisted to a repository, such as a configuration management database (CMDB). Control plane functions initiated by the Manager will coordinate the necessary orchestration components, who will choreograph endpoint data collection via defined interactions, using the Integration Service as a message broker. Instructions to perform endpoint data collection are directed to a Posture Collection Service

capable of performing collection activities utilizing any number of protocols, such as SNMP, NETCONF/RESTCONF, SCAP, SSH, WinRM, packet capture, or host-based. Instructions are orchestrated with the appropriate Posture Collection Services using serializations supported according to the collector's capabilities.

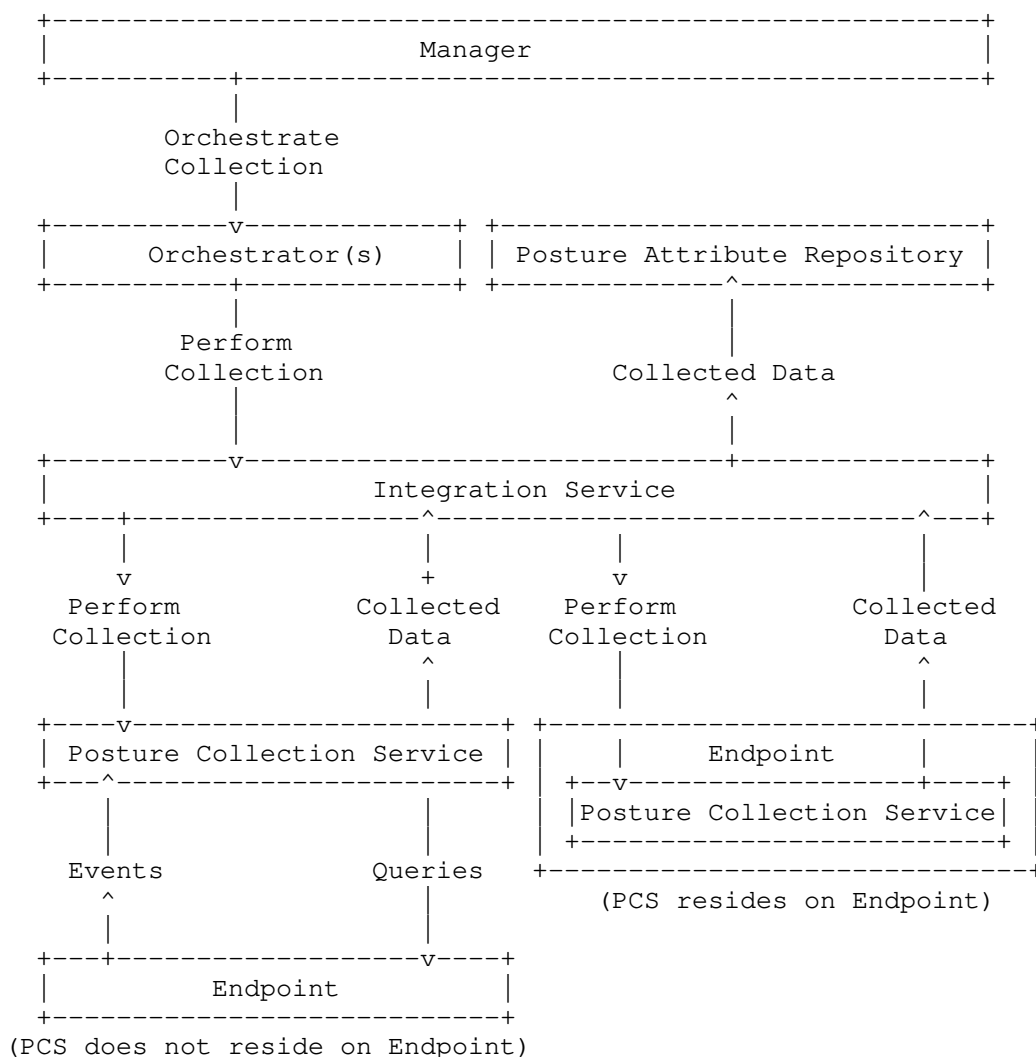


Figure 3: Decomposed Collection Sub-Architecture

4.3.1.1. Posture Collection Service

The Posture Collection Service (PCS) is a SACM component responsible for the collection of posture attributes from an endpoint or set of endpoints. A single PCS MAY be responsible for management of posture attribute collection from many endpoints. The PCS will interact with the Integration Service to receive collection instructions, and to provide collected posture attributes for persistence to one or more Posture Attribute Repositories. Collection instructions may be supplied in a variety of forms, including subscription to a publish/subscribe topic to which the Integration Service has published instructions, or via request/response-style messaging (either synchronous or asynchronous).

Four classifications of posture collections MAY be supported.

4.3.1.1.1. Ad-Hoc

Ad-Hoc collection is defined as a single collection of posture attributes, collected at a particular time. An example of ad-hoc collection is the single collection of a specific registry key.

4.3.1.1.2. Continuous/Scheduled

Continuous/Scheduled collection is defined as the ongoing, periodic collection of posture attributes. An example of scheduled collection is the collection of a specific registry key value every day at a given time.

4.3.1.1.3. Observational

This classification of collection is triggered by the observation, external to an endpoint, of information asserting posture attribute values for that endpoint. An example of observational collection is examination of netflow data for particular packet captures and/or specific information within those captures.

4.3.1.1.4. Event-based

Event-based collection may be triggered either internally or externally to the endpoint. Internal event-based collection is triggered when a posture attribute of interest is added, removed, or modified on an endpoint. This modification indicates a change in the current state of the endpoint, potentially affecting its adherence to some defined policy. Modification of the endpoint's minimum password length is an example of an attribute change which could trigger collection.

External event-based collection can be described as a collector being subscribed to an external source of information, receiving events from that external source on a periodic or continuous basis. An example of event-based collection is subscription to YANG Push notifications.

4.3.1.2. Endpoint

Building upon [I-D.ietf-sacm-terminology], the SACM Collection Sub-Architecture augments the definition of an Endpoint as a component within an organization's management domain from which a Posture Collection Service will collect relevant posture attributes.

4.3.1.3. Posture Attribute Repository

The Posture Attribute Repository is a SACM component responsible for the persistent storage of posture attributes collected via interactions between the Posture Collection Service and Endpoints.

4.3.1.4. Posture Collection Workflow

Posture collection may be triggered from a number of components, but commonly begin either via event-based triggering on an endpoint or through manual orchestration, both illustrated in Figure 3 above. Once orchestration has provided the directive to perform collection, posture collection services consume the directives. Posture collection is invoked for those endpoints overseen by the respective posture collection services. Collected data is then provided to the Integration Service, with a directive to store that information in an appropriate repository.

4.3.2. Evaluation Sub-Architecture

The Evaluation Sub-Architecture, in the SACM context, is the mechanism by which policy, expressed in the form of expected state, is compared with collected posture attributes to yield an evaluation result, that result being contextually dependent on the policy being evaluated.

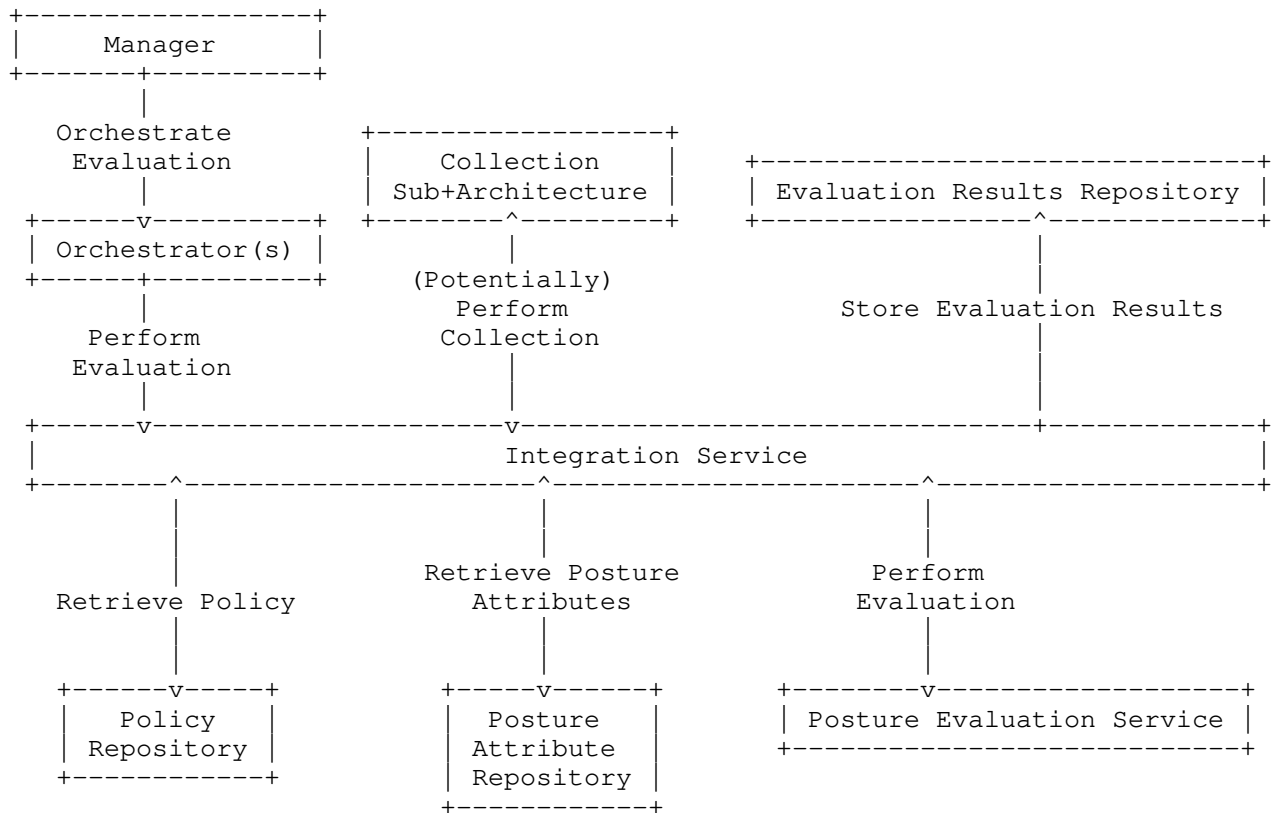


Figure 4: Decomposed Evaluation Sub-Architecture

4.3.2.1. Posture Evaluation Service

The Posture Evaluation Service (PES) represents the SACM component responsible for coordinating the policy to be evaluated and the collected posture attributes relevant to that policy, as well as the comparison engine responsible for correctly determining compliance with the expected state.

4.3.2.2. Policy Repository

The Policy Repository represents a persistent storage mechanism for the policy to be assessed against collected posture attributes to determine if an endpoint meets the desired expected state. Examples of information contained in a Policy Repository would be Vulnerability Definition Data or configuration recommendations as part of a CIS Benchmark or DISA STIG.

4.3.2.3. Evaluation Results Repository

The Evaluation Results Repository persists the information representing the results of a particular posture assessment, indicating those posture attributes collected from various endpoints which either meet or do not meet the expected state defined by the assessed policy. Consideration should be made for the context of individual results. For example, meeting the expected state for a configuration attribute indicates a correct configuration of the endpoint, whereas meeting an expected state for a vulnerable software version indicates an incorrect configuration.

4.3.2.4. Posture Evaluation Workflow

Posture evaluation is orchestrated through the Integration Service to the appropriate Posture Evaluation Service (PES). The PES will, using interactions defined by the applicable taxonomy, query both the Posture Attribute Repository and the Policy Repository to obtain relevant state data for comparison. If necessary, the PES may be required to invoke further posture collection. Once all relevant posture information has been collected, it is compared to expected state based on applicable policy. Comparison results are then persisted to an evaluation results repository for further downstream use and analysis.

5. Ecosystem Interactions

Ecosystem interactions describe the various functions between SACM components, including manager requirements, the onboarding of components, capability advertisement, administrative actions, and status updates, among others. The Manager component acts as the administrative "lead" for the SACM ecosystem, and must maintain records of registered components, manage capabilities, and more.

5.1. Manager

The Manager, being a specialized role in the architecture, enables the onboarding and capability management of the various SACM component roles. The Manager must support the set of capabilities needed to operate the SACM ecosystem.

With this in mind, the Manager must first authenticate to the Integration Service. Once authentication has succeeded, the Manager MUST establish a service handler capable of performing SACM component registration/onboarding activities (Component Registration Operation). The Manager MUST also establish a subscription to an ecosystem-wide status notification mechanism, in order to receive published status updates from other SACM components.

The following requirements exist for the Manager to establish service handlers supporting the component registration taxonomy (Component Registration Operation):

- * The Manager MUST enable the capability to receive onboarding requests,
- * The Manager MUST have the capability to generate, manage, and persist unique identifiers for all registered components,
- * The Manager MUST maintain the relationships between capabilities and payload categorizations (such as topic names or specific payload identifiers),
- * The Manager MUST have the capability to inventory and manage its "roster" (the list of registered components),
- * The Manager MUST have the capability to manage its roster's advertised capabilities, including those endpoints to which those capabilities apply.
- * In addition to supporting component registration, the Manager is responsible for many of the operational functions of the architecture, including initiating collection or evaluation, queries for repository data, or the assembly of information for downstream use.
- * The Manager MUST support making directed requests to registered components over the component's administrative interface. Administrative interface functions are described by their taxonomy, below.
- * The Manager MUST support each of the interaction categories as described above.

5.2. Component Registration

Component registration describes how an individual component becomes part of the SACM ecosystem; authenticating to the Integration Service, registering and establishing its administrative interface with, the Manager.

The component onboarding workflow involves multiple steps:

- * The component first authenticates to the Integration Service.

- * The component initiates registration with the Manager, per the component registration operation (Component Registration Operation).
- * The component handles the response from the Manager to configure a service handler allowing the component to receive directed messages over the administrative interface with the Manager.

5.3. Administrative Interface

The administrative interface represents a direct communication channel between the Manager and any registered Component. This interface allows the Manager to make directed requests to a component in order to perform specific actions.

5.3.1. Capability Advertisement Handshake

Capability Advertisement is the mechanism by which components initially indicate their capabilities to the Manager. This handshake is completed using the administrative interface with the Manager. It becomes the Manager's responsibility to persist component/capability relationships, and to provide the component the information necessary to receive and process message payloads specific to the supported capabilities.

5.3.2. Health Check

The administrative "health check" is a mechanism by which the Manager queries for the "liveness" of its roster of components, and to possibly alert users or other systems when components are no longer present. The Manager MAY enable a periodic message to each component to determine if that component is still listening to the Administrative Interface. The Health Check interaction MAY include a request for "Capability Refresh", to reinitiate the Capability Advertisement Handshake. This interaction is similar to the "Heartbeat" interaction, but is initiated by the Manager.

5.3.3. Heartbeat

The administrative "heartbeat" is a mechanism by which a Component indicates to the Manager that the Component remains connected to the ecosystem. The Heartbeat differs from the Health Check interaction in that the Component initiates the interaction, and that no response from the Manager is required.

5.3.4. Capability-specific Requests

Any number of capability-specific requests can be enabled through the administrative interface that allow the Manager to direct actions to be performed by a specific component. Utilizing the interface from a component to the Manager, this interface can be used to indicate a component has come back online, or to provide an updated capability advertisement, potentially resulting in updates to subscriptions or service handlers.

5.4. Status Notifications

A generic status notifications mechanism SHOULD be configured to which (a) the Manager is subscribed, and (b) all onboarded components can publish. Status notifications may be used by the Manager to update user interfaces, to provide notification of the start, finish, success or failure of ecosystem operations, or as events to trigger subsequent activities.

5.5. Component Interactions

Component interactions describe functionality between components relating to collection, evaluation, or other downstream processes. The following component interactions begin with the Manager providing a set of instructions to an Orchestrator or set of Orchestrators that have registered with the SACM ecosystem indicating the appropriate capabilities, such as collection or evaluation. Subscribing Orchestrator(s) MAY translate, manipulate, filter, augment, or otherwise transform the Manager's instructions into content supported through the Orchestrator's capabilities.

5.5.1. Initiate Ad-Hoc Collection

The Orchestrator supplies a payload of collection instructions to a Posture Collection Service either through direct or broadcast mechanisms. The receiving PCS components perform the required collection based on their capabilities. Each PCS then forms a payload of collected posture attributes (including endpoint identifying information) and provides that payload to the Posture Attribute Repository interface, for persistence.

5.5.2. Coordinate Periodic Collection

Similar to ad-hoc collection, the Orchestrator supplies a payload of collection instructions similar to those of ad-hoc collection. Additional information elements containing collection identification and periodicity are included.

5.5.2.1. Schedule Periodic Collection

To enable operations on periodic collection, the scheduling payload MUST include both a unique identifier for the set of collection instructions, as well as a periodicity expression to enable the collection schedule. An optional "immediate collection" flag will indicate to the collection component that, upon receipt of the collection instructions, a collection will automatically be initiated prior to engagement of the scheduled collection.

5.5.2.2. Cancel Periodic Collection

The Orchestrator disables the periodic collection of posture attributes by supplying collector(s) the unique identifier of previously scheduled collection instructions. An optional "final collection" flag will indicate to the collection component that, upon receipt of the cancellation instructions, a final ad-hoc collection is to take place.

5.5.3. Coordinate Observational/Event-based Collection

In these scenarios, the Posture Collection Service acts as the "observer". Interactions with the observer could specify a time period of observation and potentially information intended to filter observed posture attributes to aid the PCS in determining those attributes that are applicable for collection and persistence to the Posture Attribute Repository.

5.5.3.1. Initiate Observational/Event-based Collection

The Orchestrator supplies a payload of instructions to a topic or set of topics to which Posture Collection Services (observers) are subscribed. This payload could include specific instructions based on the observer's capabilities to determine specific posture attributes to observe and collect.

5.5.3.2. Cancel Observational/Event-based Collection

The Orchestrator supplies a payload of instructions to a topic or set of topics to which Posture Collection Services are subscribed. The receiving PCS components cancel the identified observational/event-based collection executing on those PCS components.

5.5.4. Persist Collected Posture Attributes

Following successful collection, Posture Collection Services (PCS) will supply the payload of collected posture attributes to the interface(s) supporting the persistent storage of those attributes to the Posture Attribute Repository. Information in this payload should include identifying information of the computing resource(s) for which attributes were collected.

5.5.5. Initiate Ad-Hoc Evaluation

The Orchestrator supplies a payload of evaluation instructions to a Posture Evaluation Services (PES) either through direct or broadcast mechanisms. The receiving PES components perform the required evaluation based on their capabilities. The PES generates a payload of posture evaluation results and publishes that payload to the Evaluation Results Repository interface, for persistence.

5.5.6. Coordinate Periodic Evaluation

Similar to ad-hoc evaluation, the Orchestrator supplies a payload of evaluation instructions similar to those of ad-hoc evaluation. Additional information elements containing evaluation identification and periodicity are included.

5.5.6.1. Schedule Periodic Evaluation

To enable operations on periodic evaluation, the scheduling payload MUST include both a unique identifier for the set of evaluation instructions, as well as a periodicity expression to enable the evaluation schedule. An optional "immediate evaluation" flag will indicate to the Posture Evaluation Service (PES) that, upon receipt of the evaluation instructions, an evaluation will automatically be initiated prior to engagement of the scheduled evaluation.

5.5.6.2. Cancel Periodic Evaluation

The Orchestrator disables the periodic evaluation of posture attributes by supplying Posture Evaluation Service(s) the unique identifier of previously scheduled evaluation instructions. An optional "final evaluation" flag will indicate to the PES that, upon receipt of the cancellation instructions, a final ad-hoc evaluation is to take place.

5.5.7. Coordinate Change-based Evaluation

A more fine-grained approach to periodic evaluation may be enabled through the triggering of Posture Evaluation based on changes to posture attribute values at the time of their collection and persistence to the Posture Attribute Repository.

5.5.7.1. Identify Attributes

The Orchestrator enables change-based evaluation through a payload published to Posture Attribute Repository component(s). This payload includes appropriate information elements describing the posture attributes on which changes in value will trigger posture evaluation.

5.5.7.2. Cancel Change-based Evaluation

An Orchestrator may disable change-based evaluation through a payload published to Posture Attribute Repository component(s), including those information elements necessary to identify those posture attributes for which change-based evaluation no longer applies.

5.5.8. Queries

Queries should allow for a "freshness" time period, allowing the requesting entity to determine if/when posture attributes must be re-collected prior to performing evaluation. This freshness time period can be "zeroed out" for the purpose of automatically triggering re-collection regardless of the most recent collection.

6. Operations

The following sections describe a number of operations required to enable a cooperative ecosystem of posture attribute collection and evaluation functions.

6.1. Component Registration

Component registration describes how an individual component becomes part of the SACM ecosystem; registering with the Manager, and establishing the administrative interface.

- * Interaction Type: Directed (Request/Response)
- * Source Component: Any component wishing to join the ecosystem, such as Posture Collection Services, Repository Interfaces, Posture Evaluation Services and more.
- * Target Component(s): Manager

6.1.1. Request Payload

When a component onboards with the ecosystem, it must identify itself to the Manager, using either descriptive information or an already-existing component unique identifier.

```
component-registration-request:
  {:component-identification:}
```

```
component-identification:
  component-unique-identifier (if re-establishing communication)
  #-OR-#
  component-type {:component-type:}
  component-name
  component-description (optional)
```

```
component-type:
  enumeration:
    - posture-collection-service
    - posture-evaluation-service
    - repository-interface
    - orchestrator
    - others?
```

When registering for the first time, the component will send identifying information including the component type and a name. If the component is re-establishing communications, for example after a restart of the component or deployment of a new version, the component only needs to supply its previously generated (and persisted) [component-unique-identifier].

6.1.2. Request Processing

When the Manager receives the component's request for onboarding, it will:

- * Generate a unique identifier, "[component-unique-identifier]", for the onboarding component,
- * Persist identifying information, including the "[component-unique-identifier]" to its component inventory, enabling an up-to-date roster of components being managed,
- * Establish the administrative interface to the onboarded component by enabling a service handler to listen for directed messages from the component.

6.1.3. Response Payload

The Manager will respond to the component with a payload including the component's unique identifier. At this point, the Manager is aware of the component's existence in the ecosystem, and the component can self-identify by virtue of receiving its unique identifier.

```
component-registration-response:  
  component-unique-identifier: [component-unique-identifier]
```

6.1.4. Response Processing

Successful receipt of the Manager's response, including the "[component-unique-identifier]", indicates the component is onboarded to the ecosystem. Using the response payload, the component can then establish it's end of the administrative interface with the Manager. The component must then persist it's unique identifier for use when re-establishing communication with the Manager after failure recovery or restart.

6.2. Administrative Interface

A number of functions may take place which, instead of being published to multiple subscribers, may require direct interaction between the Manager and a registered component (and vice-versa). During component onboarding, this direct channel, known as the Administrative Interface, is established first by the Manager and subsequently complemented by the component onboarding the SACM ecosystem. Three operations are defined for the administrative interface, but any number of application or capability-specific operations MAY be enabled using the directed messaging provided by this interface.

6.2.1. Capability Advertisement Handshake

Capability advertisement represents the ability of any registered component to inform the Manager of that component's capacity for performing certain operations. For example, a Posture Collection Service component may advertise its capability to perform collection using a particular collection system/serialization. This capability advertisement is important for the Manager to persist in order for the Manager to correctly classify components registered within the SACM ecosystem, and therefore provide the ability to publish messages to components in accordance with their capabilities.

* Interaction Type: Directed (Request/Response)

- * Source Component: Any registered component, such as Posture Collection Services, Repository Interfaces, Posture Evaluation Services and more.
- * Target Component(s): Manager

6.2.1.1. Request Payload

The component's capability advertisement request payload will include a list of "Capability URNs" (TBD IANA SECTION) that represent it's supported operational capabilities.

```
capability-advertisement:
  capabilities:
    capability-urn: [urn]
    capability-urn: [urn]
    capability-urn: [urn]
    ...
```

6.2.1.2. Request Processing

Upon receipt of the component's capability advertisement, the Manager SHOULD:

- * Persist the component's capabilities to the Manager's inventory
- * Coordinate, based on the supplied capabilities, the service handlers (for directed messages) and/or event listeners (for broadcast messages) to which the component should support.

6.2.1.3. Response Payload

The response payload delivered to the component should include the appropriate service handling/event listening information required for the component to handle further interactions based on each advertised capability. If a capability was not registered successfully, appropriate error messages SHOULD be supplied to inform the component of the failure(s).

```
capability-advertisement-response:
  capabilities:
    capability:
      capability-urn: [urn]
      registration-status: (success | failure)
      service-handler-or-event-listener: [info]
      messages: [messages]
    capability:
      capability-urn: [urn]
      registration-status: (success | failure)
      service-handler-or-event-listener: [info]
      messages: [messages]
```

6.2.1.4. Response Processing

Once the component has received the response to its capability advertisement, it should configure the capability-specific service handler(s) or event listener(s). Once these handlers/listeners have been configured, the component is considered fully onboarded to the SACM ecosystem.

6.2.2. Health Check

As time passes, it is important that the Manager maintains knowledge of all registered component's current operational status. The health check operation describes the efforts taken by the Manager to maintain the most up-to-date inventory of it's component roster, and to potentially trigger events to users or outside systems (e.g. a SIEM or SOAR) indicating unavailable components.

- * Interaction Type: Directed (Request/Response)
- * Source Component: Manager
- * Target Component(s): Any registered component, such as Posture Collection Services, Repository Interfaces, Posture Evaluation Services and more.

6.2.2.1. Request Payload

The request for the health check is a simple "ping".

```
health-check-request:
  action: ping
```

6.2.2.2. Request Processing

When the target component receives the health check request, the target component need only respond that it is operational and connected to the integration service. This is a simple "Hello component, are you listening? Yes, I am" interaction. The health check request from the Manager should be made with an appropriately small timeout indicator; only an operational component will be able to respond to the request, so if that component is offline and cannot respond, the Manager should not be kept waiting for an extended amount of time.

6.2.2.3. Response Payload

When responding to the health check request, the response payload will simply indicate success: ~~~~~ health-check-response: response: success ~~~~~

6.2.2.4. Response Processing

Upon receipt of the "health-check-response" payload, the Manager will update its inventory of currently operational components with the timestamp of the receipt. Manager implementations may raise alerts, inform users, or take other actions when health checks are unsuccessful, at their discretion.

6.2.3. Heartbeat

As time passes and SACM ecosystem components which have previously registered are brought offline (perhaps for maintenance or redeployment) and back online, it is important that registered components maintain administrative contact with the Manager. The heartbeat operation describes the efforts taken by a registered component to determine the status of contact with the Manager, and to take appropriate action if such contact cannot be made.

- * Interaction Type: Directed (Request/Response)
- * Source Component: Any registered component, such as Posture Collection Services, Repository Interfaces, Posture Evaluation Services and more.
- * Target Component(s): Manager

6.2.3.1. Request Payload

The request payload simply defines the heartbeat action:

```
heartbeat-request:
  action: pulse
```

6.2.3.2. Request Processing

When the Manager receives the heartbeat request, it need only respond that it is operational and connected to the integration service. This is a simple "Hello Manager, are you listening? Yes, I am" interaction. The heartbeat request from the component should be made with an appropriately small timeout indicator; only an operational Manager will be able to respond to the request, so if it is offline and cannot respond, the component should not be kept waiting for an extended amount of time.

6.2.3.3. Response Payload

When responding to the heartbeat request, the response payload will simply indicate success: ~~~~~ heartbeat-response: response: success ~~~~~

6.2.3.4. Response Processing

Upon receipt of the "heartbeat-response" payload, the component may reset its heartbeat timer and continue normal operations, awaiting incoming message payloads. Component implementations may raise alerts, inform users, or take other actions when heartbeat requests are unsuccessful (potentially indicating a downed Manager), at their discretion.

6.3. Status Notification

From time to time during the performance of any given operation, a component may need to supply status information to the Manager (or any other concerned component), for use in display to users, or to trigger other events within the SACM ecosystem. The status notification operation is designed to allow any component to broadcast status message payloads to any subscribers with the need to know. For example, a collection component could broadcast to the Manager that it has initiated collection, subsequent collection progress updates, and finally completion or error conditions.

- * Interaction Type: Broadcast (Publish/Subscribe)
- * Source Component: Any registered component, such as Posture Collection Services, Repository Interfaces, Posture Evaluation Services and more.

- * Target Component(s): Typically the Manager, but any registered component may subscribe to status notifications.

6.3.1. Request Payload

At a minimum, the payload broadcast for a status notification MUST include the status message and the publishing component's "component-unique-identifier". Further identifying information, such as status codes, operation indicators, etc., MAY be provided by implementing components.

```
status-notification:
  publisher: [component-unique-identifier]
  message: [message]
  [additional information]
```

6.3.2. Request Processing

When subscribers are notified of the status message, respective components may act upon them in component/application-specific ways, including persisting those messages to repositories, forwarding to log aggregation tools, displaying on user interfaces, and so on. Potential for use of component status notifications is only limited by application implementations.

6.3.3. Response Payload

N/A

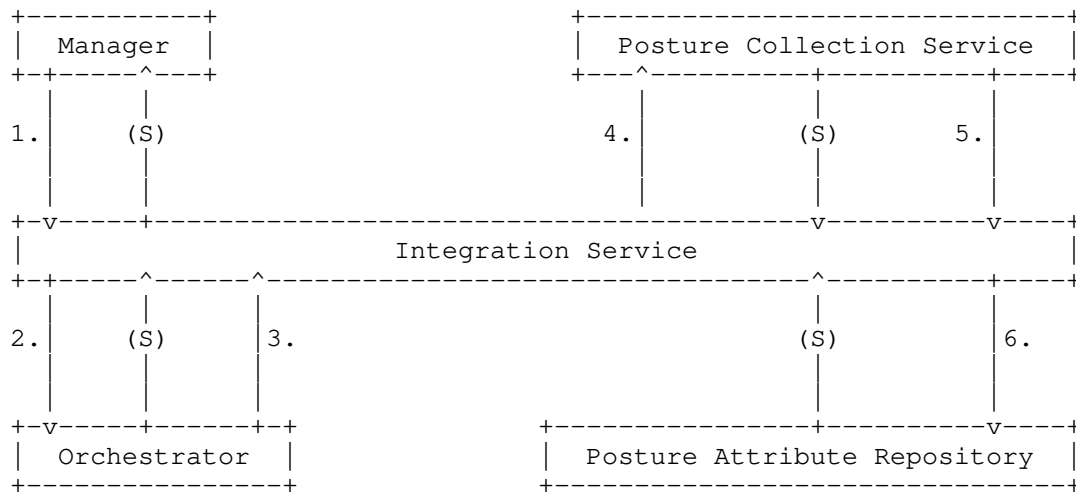
6.3.4. Response Processing

N/A

6.4. Initiate Ad-Hoc Collection

The Ad-hoc collection workflow MAY be initiated by the Manager, via user interaction, or through a Posture Evaluation Service, and represents a single, point-in-time operation to collect posture attributes from applicable endpoints. The SACM Producer initiates a message payload, either through directed channels (such as the administrative interface) or through broadcast notifications to multiple subscribers, to Orchestrator components. Orchestrators MAY manipulate the Manager's collection instructions according to various collection capabilities, prior to providing those instructions to Posture Collection Service (PCS) components. Once collection instructions are received by the PCS, it will collect the requested posture attributes from the designated endpoints, using its advertised collection capabilities. The following diagram

illustrates this workflow with the Manager as the initiating SACM Producer:



1. The Manager initiates a request to one or more Orchestrators to perform collection,
2. The Orchestrator receives collection instructions and potentially manipulates them according to one or more collection capabilities,
3. The Orchestrator publishes a notification to subscribed Posture Collection Service components, indicating the posture attributes to be collected,
4. The Posture Collection Service receives the collection instructions and performs the actual collection of posture attributes from an endpoint or endpoints.
5. The Posture Collection Service publishes a notification(s) containing the collected posture attributes to be persisted to the Posture Attribute Repository,
6. The Posture Attribute Repository persists the collected posture attributes, potentially performing normalization of the data as part of its process.

Interactions labeled (S) indicate the capability of each component to publish status notifications, subscribed to by the Manager.

6.4.1. SACM Producer to Orchestrator

The Ad-hoc collection workflow MAY be initiated by a number of SACM components, such as the Manager, a Posture Evaluation Service, or other events outside the scope of this document.

- * Interaction Type: Directed (Request/Response) or Broadcast (Publish/Subscribe)
- * Source Component: Various
- * Target Component(s): Orchestrator

6.4.1.1. Request Payload

A request to orchestrate posture attribute collection MUST include enough information to describe those attributes being collected, and MAY include endpoint targeting information.

collection-instructions:
TBD

6.4.1.2. Request Processing

When the Orchestrator receives the collection instructions, it may be required to manipulate them according to the capabilities it's collector(s) support. For example, generic collection instructions could be transformed to the appropriate OVAL serialization for collection via OVAL-compliant collectors.

6.4.1.3. Response Payload

Orchestrators have the option to provide broadcast status update messages to indicate success, failure, or other error messages when receiving posture collection orchestration payloads.

6.4.1.4. Response Processing

N/A

6.4.2. Orchestrator to Posture Collection Service

Once the Orchestrator has received collection instructions from the initiating SACM component, and has performed any manipulation of the instructions to conform to it's capabilities, it will provide those instructions to relevant Posture Collection Services.

- * Interaction Type: Directed (Request/Response) or Broadcast (Publish/Subscribe)
- * Source Component: Orchestrator
- * Target Component(s): Posture Collection Service

6.4.2.1. Request Payload

The payload exchanged between the Orchestrator and it's associated Posture Collection Services will be collection instructions adhering to a data model supported by the PCS based on its advertised capabilities.

collection-instructions:
TBD

6.4.2.2. Request Processing

Upon receipt of the payload containing collection instructions, the Posture Collection Service should parse and validate them, indicating any errors in the process. If the payload does not conform to any serialization or data model to which the PCS' capabilities correspond, status messages indicating such nonconformance SHOULD be provided to both the Orchestrator and the initiating SACM producer.

Once successfully parsed and validated, the PCS MUST perform collection of posture attributes according to the collection instructions, from any endpoint to which the PCS has access, or from the list of endpoints described in any targeting information included in the collection instructions.

6.4.2.3. Response Payload

Posture Collection Service components will respond using the generic status update mechanisms to indicate success, failure, or any errors that occur. Errors may occur parsing collection instructions, verifying them, targeting indicated endpoints, or from the act of collecting the indicated posture attributes.

6.4.2.4. Response Processing

Any messages received by components regarding the success, failure, or errors involved in the collection of posture attributes MAY be processed according to the receiving components' capabilities.

6.4.3. Posture Collection Service to Posture Attribute Repository

Upon completion of posture attribute collection, the PCS constructs the payload of collected attributes based on its advertised capabilities, e.g. OVAL system characteristics. This payload is provided to either a specific posture attribute repository via directed messages or to subscribed repository interfaces via broadcast messages.

- * Interaction Type: Directed (Request/Response) or Broadcast (Publish/Subscribe)
- * Source Component: Posture Collection Service
- * Target Component(s): Posture Attribute Repository

6.4.3.1. Request Payload

The payload supplied by the Posture Collection Service SHOULD conform to information and data models supported by its advertised capabilities. These data models, at a minimum, SHOULD include name/value pairs for each collected attribute.

```
collection-results:
  [
    attribute-name,
    attribute-value
  ]
```

6.4.3.2. Request Processing

As the Posture Attribute Repository interface receives the payload of collected posture attributes, some data normalization MAY occur in order to persist the information most efficiently based on the persistence technology. This normalization is dependent on the implementation of the repository interface as well as the persistence technology. For example, OVAL system characteristics, an XML payload, could be normalized to a property graph representation when persisted to a Neo4j database.

6.4.3.3. Response Payload

Once the Posture Attribute Repository has received, it MAY respond to the Posture Collection Service that it has successfully received the collected posture attributes. This response would only be applicable when receiving payloads via directed requests. If payloads are received via broadcast interactions, there may not be a PCS to which a response can be sent. The Posture Attribute Repository MAY utilize

the generic status update interactions to provide response messages to appropriate subscribers.

6.4.3.4. Response Processing

Any messages received by components regarding the success, failure, or errors involved in the persistence of collected posture attributes MAY be processed according to the receiving components' capabilities. For example, a generic status update message could be processed by a Manager component, correlated with the initial posture collection instructions in order to "close the loop" on the posture attribute collection workflow.

6.5. Initiate Ad-Hoc Evaluation

Manager to Orchestrator ### Orchestrator to Evaluator ###
Evaluator to Posture Evaluation Repository

7. Privacy Considerations

[TBD]

8. Security Considerations

[TBD]

9. IANA Considerations

[TBD] Some boilerplate code...

9.1. Component Types

URI: "urn:ietf:sacm:component-type" Description: The allowed enumeration of the various component types permitted to utilize the SACM ecosystem.

- * Manager
- * Orchestrator
- * Collector
- * Evaluator
- * Repository Interface
- * [MORE]

9.2. Component Capabilities

Health Check A URN representing a component's capability to initiate Health Check operations and to process any provided response payloads.

URN: "urn:ietf:sacm:capability:action:health-check"

9.2.1. Heartbeat

A URN representing a component's capability to initiate Heartbeat operations and to process any provided response payloads.

URN: "urn:ietf:sacm:capability:action:heartbeat"

9.2.2. Status Notification (Publish)

A URN representing a component's capability to publish status notifications.

URN: "urn:ietf:sacm:capability:publish:status-notification"

9.2.3. Status Notification (Subscribe)

A URN representing a component's capability to subscribe to status notification events.

URN: "urn:ietf:sacm:capability:subscribe:status-notification"

10. References

10.1. Normative References

- [I-D.ietf-sacm-ecp] Haynes, D., Fitzgerald-McKay, J., and L. Lorenzin, "Endpoint Posture Collection Profile", Work in Progress, Internet-Draft, draft-ietf-sacm-ecp-05, 21 June 2019, <<https://www.ietf.org/archive/id/draft-ietf-sacm-ecp-05.txt>>.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<https://www.rfc-editor.org/info/rfc2119>>.
- [RFC6120] Saint-Andre, P., "Extensible Messaging and Presence Protocol (XMPP): Core", RFC 6120, DOI 10.17487/RFC6120, March 2011, <<https://www.rfc-editor.org/info/rfc6120>>.

- [RFC8412] Schmidt, C., Haynes, D., Coffin, C., Waltermire, D., and J. Fitzgerald-McKay, "Software Inventory Message and Attributes (SWIMA) for PA-TNC", RFC 8412, DOI 10.17487/RFC8412, July 2018, <<https://www.rfc-editor.org/info/rfc8412>>.
- [RFC8600] Cam-Winget, N., Ed., Appala, S., Pope, S., and P. Saint-Andre, "Using Extensible Messaging and Presence Protocol (XMPP) for Security Information Exchange", RFC 8600, DOI 10.17487/RFC8600, June 2019, <<https://www.rfc-editor.org/info/rfc8600>>.

10.2. Informative References

- [CISCONTROLS] "CIS Controls v7.1", n.d., <<https://www.cisecurity.org/controls>>.
- [draft-birkholz-sacm-yang-content] Birkholz, H. and N. Cam-Winget, "YANG subscribed notifications via SACM Statements", n.d., <<https://tools.ietf.org/html/draft-birkholz-sacm-yang-content-01>>.
- [HACK100] "IETF 100 Hackathon - Vulnerability Scenario EPCP+XMPP", n.d., <<https://www.github.com/sacmwg/vulnerability-scenario/ietf-hackathon>>.
- [HACK101] "IETF 101 Hackathon - Configuration Assessment XMPP", n.d., <<https://www.github.com/CISecurity/Integration>>.
- [HACK102] "IETF 102 Hackathon - YANG Collection on Traditional Endpoints", n.d., <<https://www.github.com/CISecurity/YANG>>.
- [HACK103] "IETF 103 Hackathon - N/A", n.d., <<https://www.ietf.org/how/meetings/103/>>.
- [HACK104] "IETF 104 Hackathon - A simple XMPP client", n.d., <<https://github.com/CISecurity/SACM-Architecture>>.
- [HACK105] "IETF 105 Hackathon - A more robust XMPP client including collection extensions", n.d., <<https://github.com/CISecurity/SACM-Architecture>>.
- [HACK99] "IETF 99 Hackathon - Vulnerability Scenario EPCP", n.d., <<https://www.github.com/sacmwg/vulnerability-scenario/ietf-hackathon>>.

[I-D.ietf-i2nsf-terminology]

Hares, S., Strassner, J., Lopez, D. R., Xia, L., and H. Birkholz, "Interface to Network Security Functions (I2NSF) Terminology", Work in Progress, Internet-Draft, draft-ietf-i2nsf-terminology-08, 5 July 2019, <<https://www.ietf.org/archive/id/draft-ietf-i2nsf-terminology-08.txt>>.

[I-D.ietf-sacm-terminology]

Birkholz, H., Lu, J., Strassner, J., Cam-Winget, N., and A. Montville, "Security Automation and Continuous Monitoring (SACM) Terminology", Work in Progress, Internet-Draft, draft-ietf-sacm-terminology-16, 14 December 2018, <<https://www.ietf.org/archive/id/draft-ietf-sacm-terminology-16.txt>>.

[MQTT] "MQTT", n.d., <<https://mqtt.org/mqtt-specification/>>.

[NIST800126]

Waltermire, D., Quinn, S., Booth, H., Scarfone, K., and D. Prisaca, "SP 800-126 Rev. 3 - The Technical Specification for the Security Content Automation Protocol (SCAP) - SCAP Version 1.3", February 2018, <<https://csrc.nist.gov/publications/detail/sp/800-126/rev-3/final>>.

[NISTIR7694]

Halbardier, A., Waltermire, D., and M. Johnson, "NISTIR 7694 Specification for Asset Reporting Format 1.1", n.d., <<https://csrc.nist.gov/publications/detail/nistir/7694/final>>.

[RFC3444] Pras, A. and J. Schoenwaelder, "On the Difference between Information Models and Data Models", RFC 3444, DOI 10.17487/RFC3444, January 2003, <<https://www.rfc-editor.org/info/rfc3444>>.

[RFC4949] Shirey, R., "Internet Security Glossary, Version 2", FYI 36, RFC 4949, DOI 10.17487/RFC4949, August 2007, <<https://www.rfc-editor.org/info/rfc4949>>.

[RFC5023] Gregorio, J., Ed. and B. de hOra, Ed., "The Atom Publishing Protocol", RFC 5023, DOI 10.17487/RFC5023, October 2007, <<https://www.rfc-editor.org/info/rfc5023>>.

- [RFC5209] Sangster, P., Khosravi, H., Mani, M., Narayan, K., and J. Tardo, "Network Endpoint Assessment (NEA): Overview and Requirements", RFC 5209, DOI 10.17487/RFC5209, June 2008, <<https://www.rfc-editor.org/info/rfc5209>>.
- [RFC6192] Dugal, D., Pignataro, C., and R. Dunn, "Protecting the Router Control Plane", RFC 6192, DOI 10.17487/RFC6192, March 2011, <<https://www.rfc-editor.org/info/rfc6192>>.
- [RFC7632] Waltermire, D. and D. Harrington, "Endpoint Security Posture Assessment: Enterprise Use Cases", RFC 7632, DOI 10.17487/RFC7632, September 2015, <<https://www.rfc-editor.org/info/rfc7632>>.
- [RFC8248] Cam-Winget, N. and L. Lorenzin, "Security Automation and Continuous Monitoring (SACM) Requirements", RFC 8248, DOI 10.17487/RFC8248, September 2017, <<https://www.rfc-editor.org/info/rfc8248>>.
- [RFC8322] Field, J., Banghart, S., and D. Waltermire, "Resource-Oriented Lightweight Information Exchange (ROLIE)", RFC 8322, DOI 10.17487/RFC8322, February 2018, <<https://www.rfc-editor.org/info/rfc8322>>.
- [XMPPEXT] "XMPP Extensions", n.d., <<https://xmpp.org/extensions/>>.

Appendix A. Security Domain Workflows

This section describes three primary information security domains from which workflows may be derived: IT Asset Management, Vulnerability Management, and Configuration Management.

A.1. IT Asset Management

Information Technology asset management is easier said than done. The [CISCONTROLS] have two controls dealing with IT asset management. Control 1, Inventory and Control of Hardware Assets, states, "Actively manage (inventory, track, and correct) all hardware devices on the network so that only authorized devices are given access, and unauthorized and unmanaged devices are found and prevented from gaining access." Control 2, Inventory and Control of Software Assets, states, "Actively manage (inventory, track, and correct) all software on the network so that only authorized software is installed and can execute, and that unauthorized and unmanaged software is found and prevented from installation or execution."

In spirit, this covers all of the processing entities on your network (as opposed to things like network cables, dongles, adapters, etc.), whether physical or virtual, on-premises or in the cloud.

A.1.1. Components, Capabilities and Workflow(s)

TBD

A.1.1.1. Components

TBD

A.1.1.2. Capabilities

An IT asset management capability needs to be able to:

- * Identify and catalog new assets by executing Target Endpoint Discovery Tasks
- * Provide information about its managed assets, including uniquely identifying information (for that enterprise)
- * Handle software and/or hardware (including virtual assets)
- * Represent cloud hybrid environments

A.1.1.3. Workflow(s)

TBD

A.2. Vulnerability Management

Vulnerability management is a relatively established process. To paraphrase the [CISCONTROLS], continuous vulnerability management is the act of continuously acquiring, assessing, and taking subsequent action on new information in order to identify and remediate vulnerabilities, therefore minimizing the window of opportunity for attackers.

A vulnerability assessment (i.e. vulnerability detection) is performed in two steps:

- * Endpoint information collected by the endpoint management capabilities is examined by the vulnerability management capabilities through Evaluation Tasks.

- * If the data possessed by the endpoint management capabilities is insufficient, a Collection Task is triggered and the necessary data is collected from the target endpoint.

Vulnerability detection relies on the examination of different endpoint information depending on the nature of a specific vulnerability. Common endpoint information used to detect a vulnerability includes:

- * A specific software version is installed on the endpoint
- * File system attributes
- * Specific state attributes

In some cases, the endpoint information needed to determine an endpoint's vulnerability status will have been previously collected by the endpoint management capabilities and available in a Repository. However, in other cases, the necessary endpoint information will not be readily available in a Repository and a Collection Task will be triggered to perform collection from the target endpoint. Of course, some implementations of endpoint management capabilities may prefer to enable operators to perform this collection even when sufficient information can be provided by the endpoint management capabilities (e.g. there may be freshness requirements for information).

A.2.1. Components, Capabilities and Workflow(s)

TBD

A.2.1.1. Components

TBD

A.2.1.2. Capabilities

TBD

A.2.1.3. Workflow(s)

TBD

A.3. Configuration Management

Configuration management involves configuration assessment, which requires state assessment. The [CISCONTROLS] specify two high-level controls concerning configuration management (Control 5 for non-network devices and Control 11 for network devices). As an aside, these controls are listed separately because many enterprises have different organizations for managing network infrastructure and workload endpoints. Merging the two controls results in the following paraphrasing: Establish, implement, and actively manage (track, report on, correct) the security configuration of systems using a rigorous configuration management and change control process in order to prevent attackers from exploiting vulnerable services and settings.

Typically, an enterprise will use configuration guidance from a reputable source, and from time to time they may tailor the guidance from that source prior to adopting it as part of their enterprise standard. The enterprise standard is then provided to the appropriate configuration assessment tools and they assess endpoints and/or appropriate endpoint information.

A preferred flow follows:

- * Reputable source publishes new or updated configuration guidance
- * Enterprise configuration assessment capability retrieves configuration guidance from reputable source
- * Optional: Configuration guidance is tailored for enterprise-specific needs
- * Configuration assessment tool queries asset inventory repository to retrieve a list of affected endpoints
- * Configuration assessment tool queries configuration state repository to evaluate compliance
- * If information is stale or unavailable, configuration assessment tool triggers an ad hoc assessment

The SACM architecture needs to support varying deployment models to accommodate the current state of the industry, but should strongly encourage event-driven approaches to monitoring configuration.

A.3.1. Components, Capabilities and Workflow(s)

This section provides more detail about the components and capabilities required when considering the aforementioned configuration management workflow.

A.3.1.1. Components

The following is a minimal list of SACM Components required to implement the aforementioned configuration assessment workflow.

- * Configuration Policy Feed: An external source of authoritative configuration recommendations.
- * Configuration Policy Repository: An internal repository of enterprise standard configurations.
- * Configuration Assessment Orchestrator: A component responsible for orchestrating assessments.
- * Posture Attribute Collection Subsystem: A component responsible for collection of posture attributes from systems.
- * Posture Attribute Repository: A component used for storing system posture attribute values.
- * Configuration Assessment Evaluator: A component responsible for evaluating system posture attribute values against expected posture attribute values.
- * Configuration Assessment Results Repository: A component used for storing evaluation results.

A.3.1.2. Capabilities

Per [RFC8248], solutions MUST support capability negotiation. Components implementing specific interfaces and operations (i.e. interactions) will need a method of describing their capabilities to other components participating in the ecosystem; for example, "As a component in the ecosystem, I can assess the configuration of Windows, MacOS, and AWS using OVAL".

A.3.1.3. Configuration Assessment Workflow

This section describes the components and interactions in a basic configuration assessment workflow. For simplicity, error conditions are recognized as being necessary and are not depicted. When one component messages another component, the message is expected to be handled appropriately unless there is an error condition, or other notification, messaged in return.

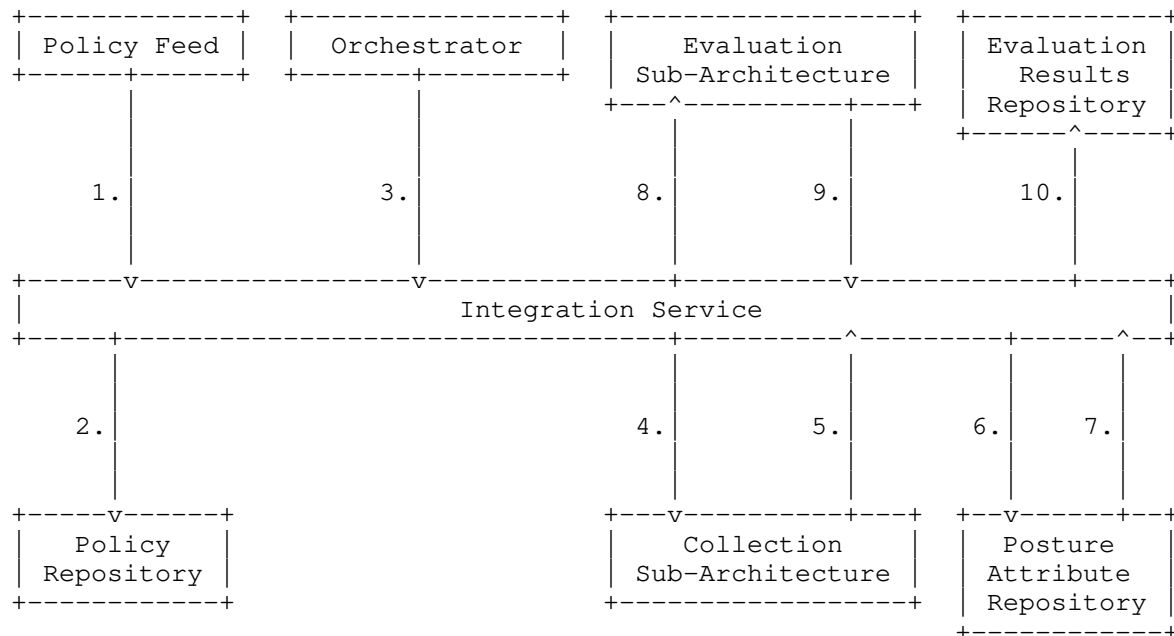


Figure 5: Configuration Assessment Component Interactions

Figure 5 depicts configuration assessment components and their interactions, which are further described below.

1. A policy feed provides a configuration assessment policy payload to the Integration Service.
2. The Policy Repository, a consumer of Policy Feed information, receives and persists the Policy Feed's payload.
3. Orchestration component(s), either manually invoked, scheduled, or event-based, publish a payload to begin the configuration assessment process.

4. If necessary, Collection Sub-Architecture components may be invoked to collect needed posture attribute information.
5. If necessary, the Collection Sub-Architecture will provide collected posture attributes to the Integration Service for persistence to the Posture Attribute Repository.
6. The Posture Attribute Repository will consume a payload querying for relevant posture attribute information.
7. The Posture Attribute Repository will provide the requested information to the Integration Service, allowing further orchestration payloads requesting the Evaluation Sub-Architecture perform evaluation tasks.
8. The Evaluation Sub-Architecture consumes the evaluation payload and performs component-specific state comparison operations to produce evaluation results.
9. A payload containing evaluation results are provided by the Evaluation Sub-Architecture to the Integration Service
10. Evaluation results are consumed by/persisted to the Evaluation Results Repository

In the above flow, the payload information is expected to convey the context required by the receiving component for the action being taken under different circumstances. For example, a directed message sent from an Orchestrator to a Collection sub-architecture might be telling that Collector to watch a specific posture attribute and report only specific detected changes to the Posture Attribute Repository, or it might be telling the Collector to gather that posture attribute immediately. Such details are expected to be handled as part of that payload, not as part of the architecture described herein.

Authors' Addresses

Adam W. Montville
Center for Internet Security
31 Tech Valley Drive
East Greenbush, NY 12061
United States of America

Email: adam.montville.sdo@gmail.com

Bill Munyan
Center for Internet Security
31 Tech Valley Drive
East Greenbush, NY 12061
United States of America

Email: bill.munyan.ietf@gmail.com

SACM Working Group
Internet-Draft
Intended status: Standards Track
Expires: 8 September 2022

H. Birkholz
Fraunhofer SIT
J. Fitzgerald-McKay
National Security Agency
C. Schmidt
The MITRE Corporation
D. Waltermire
NIST
7 March 2022

Concise Software Identification Tags
draft-ietf-sacm-coswid-21

Abstract

ISO/IEC 19770-2:2015 Software Identification (SWID) tags provide an extensible XML-based structure to identify and describe individual software components, patches, and installation bundles. SWID tag representations can be too large for devices with network and storage constraints. This document defines a concise representation of SWID tags: Concise SWID (CoSWID) tags. CoSWID supports a similar set of semantics and features as SWID tags, as well as new semantics that allow CoSWIDs to describe additional types of information, all in a more memory efficient format.

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 8 September 2022.

Copyright Notice

Copyright (c) 2022 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 Revised BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Revised BSD License.

Table of Contents

1.	Introduction	3
1.1.	The SWID and CoSWID Tag Lifecycle	4
1.2.	Concise SWID Format	8
1.3.	Requirements Notation	8
2.	Concise SWID Data Definition	8
2.1.	Character Encoding	10
2.2.	Concise SWID Extensions	10
2.3.	The concise-swid-tag Map	13
2.4.	concise-swid-tag Co-Constraints	18
2.5.	The global-attributes Group	18
2.6.	The entity-entry Map	19
2.7.	The link-entry Map	21
2.8.	The software-meta-entry Map	25
2.9.	The Resource Collection Definition	28
2.9.1.	The hash-entry Array	28
2.9.2.	The resource-collection Group	28
2.9.3.	The payload-entry Map	32
2.9.4.	The evidence-entry Map	32
2.10.	Full CDDL Specification	33
3.	Determining the Type of CoSWID	39
4.	CoSWID Indexed Label Values	40
4.1.	Version Scheme	40
4.2.	Entity Role Values	42
4.3.	Link Ownership Values	44
4.4.	Link Rel Values	44
4.5.	Link Use Values	46
5.	URI Schemes	47
5.1.	"swid" URI Scheme	47
5.2.	"swidpath" URI Scheme	48
6.	IANA Considerations	49
6.1.	CoSWID Items Registry	49
6.2.	Software Tag Values Registries	52
6.2.1.	Registration Procedures	52
6.2.2.	Private Use of Index and Name Values	52
6.2.3.	Expert Review Criteria	53
6.2.4.	Software Tag Version Scheme Values Registry	53
6.2.5.	Software Tag Entity Role Values Registry	55

6.2.6. Software Tag Link Ownership Values Registry	56
6.2.7. Software Tag Link Relationship Values Registry	57
6.2.8. Software Tag Link Use Values Registry	60
6.3. swid+cbor Media Type Registration	61
6.4. CoAP Content-Format Registration	62
6.5. CBOR Tag Registration	62
6.6. URI Scheme Registrations	62
6.6.1. URI-scheme swid	63
6.6.2. URI-scheme swidpath	63
6.7. CoSWID Model for use in SWIMA Registration	64
7. Signed CoSWID Tags	64
8. CBOR-Tagged CoSWID Tags	67
9. Security Considerations	67
10. Privacy Consideration	71
11. Change Log	72
12. References	77
12.1. Normative References	77
12.2. Informative References	80
Acknowledgments	81
Contributors	81
Authors' Addresses	82

1. Introduction

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 software, a patch, or an installation bundle, which are referred to as software components in this document. 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.

SWID tags are used to support a number of processes including but not limited to:

- * Software Inventory Management, a part of a 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 Attester-produced event logs that complement attestation Evidence [I-D.ietf-rats-architecture].

While there are very few required fields in SWID tags, there are many optional fields that support different uses. A SWID tag consisting of only required fields might be a few hundred bytes in size; however, 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 usage scenarios, 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 through the use of the Concise Binary Object Representation (CBOR) [RFC8949].

Size comparisons between XML SWID and CoSWID mainly depend on domain-specific applications and the complexity of attributes used in instances. While the values stored in CoSWID are often unchanged and therefore not reduced in size compared to an XML SWID, the scaffolding that the CoSWID encoding represents is significantly smaller by taking up 10 percent or less in size. This effect is visible in representation sizes, which in early experiments benefited from a 50 percent to 85 percent reduction in generic usage scenarios. Additional size reduction is enabled with respect to the memory footprint of XML parsing/validation.

In a CoSWID, the human-readable labels of SWID data items are replaced with more concise integer labels (indices). This approach allows SWID and CoSWID to share a common implicit information model, with CoSWID providing an alternate data model [RFC3444]. While SWID and CoSWID are intended to share the same implicit information model, this specification does not define this information model, or a mapping between the two data formats. While an attempt to align SWID and CoSWID tags has been made here, future revisions of ISO/IEC 19770-2:2015 or this specification might cause this implicit information model to diverge, since these specifications are maintained by different standards groups.

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 and CoSWID 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 software component's SWID tag is also installed. Likewise, when the software component 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 SWID specification and supporting guidance provided in NIST Internal Report (NISTIR) 8060: Guidelines for the Creation of Interoperable SWID Tags [SWID-GUIDANCE] defines four types of SWID tags: primary, patch, corpus, and supplemental. The following text is paraphrased from these sources.

1. Primary Tag - A SWID or CoSWID tag that identifies and describes an installed software component on an endpoint. A primary tag is intended to be installed on an endpoint along with the corresponding software component.
2. Patch Tag - A SWID or CoSWID tag that identifies and describes an installed patch that has made incremental changes to a software component installed on an endpoint. A patch tag is intended to be installed on an endpoint along with the corresponding software component patch.
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 allows tools and users to record their own metadata about a software component without modifying CoSWID primary or patch tags created by a software provider.

The type of a tag is determined by specific data elements, which are discussed in Section 3, which also provides normative language for CoSWID semantics that implement this lifecycle. The following information helps to explain how these semantics apply to use of a CoSWID tag.

Corpus, primary, and patch tags have similar functions in that they describe the existence and/or presence of different types of software components (e.g., software installers, software installations, software patches), and, potentially, different states of these software components. Supplemental tags have the same structure as other tags, but are used to provide information not contained in the referenced corpus, primary, and 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.

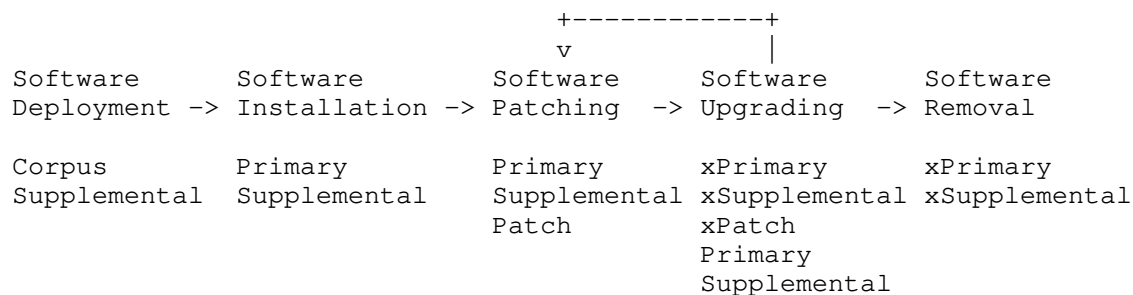


Figure 1: Use of Tag Types in the Software Lifecycle

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. A detailed description of the four tags types is provided in Section 2.3. The figure identifies the types of tags that are used in each lifecycle event.

There are many ways in which software tags might be managed for the host the software is installed on. For example, software tags could be made available on the host or to an external software manager when storage is limited on the host.

In these cases the host or external software manager is responsible for management of the tags, including deployment and removal of the tags as indicated by the above lifecycle. Tags are deployed and previously deployed tags that are typically removed (indicated by an "x" prefix) at each lifecycle stage, 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).

Corpus tags are not actually deployed on the target system but are intended to support deployment procedures and their dependencies at install-time, such as to verify the installation media.

- 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 can also be installed during software installation to provide information about software fixes deployed along with the base software installation.

- Software Patching. A new patch tag is provided, when a patch is applied to the software component, 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.

Note: In the context of software tagging software patching and updating differ in an important way. When installing a patch, a set of file modifications are made to pre-installed software which do not alter the version number or the descriptive metadata of an installed software component. An update can also make a set of file modifications, but the version number or the descriptive metadata of an installed software component are changed.

- 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.

As 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.

Understanding the use of CoSWIDs in the software lifecycle provides a basis for understanding the information provided in a CoSWID and the associated semantics of this information. 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 Format

This document defines the CoSWID tag format, which is based on CBOR. CBOR-based CoSWID tags offer a more concise representation of SWID information as compared to the XML-based SWID tag representation in ISO-19770-2:2015. The structure of a CoSWID is described via the Concise Data Definition Language (CDDL) [RFC8610]. The resulting CoSWID data definition is aligned to the information able to be expressed with the XML schema definition of ISO-19770-2:2015 [SWID]. This alignment allows both SWID and CoSWID tags to represent a common set of software component information and allows CoSWID tags to support the same uses as a SWID tag.

The vocabulary, i.e., the CDDL names of the types and members used in the CoSWID CDDL specification, are mapped to more concise labels represented as small integer values (indices). The names used in the CDDL specification and the mapping to the CBOR representation using integer indices is based on the vocabulary of the XML attribute and element names defined in ISO/IEC 19770-2:2015.

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

2. Concise SWID Data Definition

The following describes the general rules and processes for encoding data using CDDL representation. Prior familiarity with CBOR and CDDL concepts will be helpful in understanding this CoSWID specification.

This section describes the conventions by which a CoSWID is represented in the CDDL structure. The CamelCase [CamelCase] notation used in the XML schema definition is changed to a hyphen-separated notation [KebabCase] (e.g., ResourceCollection is named resource-collection) in the CoSWID CDDL specification. This deviation from the original notation used in the XML representation reduces ambiguity when referencing certain attributes in corresponding textual descriptions. An attribute referred to by its name in CamelCase notation explicitly relates to XML SWID tags; an attribute referred to by its name in KebabCase notation explicitly relates to CBOR CoSWID tags. This approach simplifies the composition of further work that reference both XML SWID and CBOR CoSWID documents.

In most cases, mapping attribute names between SWID and CoSWID can be done automatically by converting between CamelCase and KebabCase attribute names. However, some CoSWID CDDL attribute names show greater variation relative to their corresponding SWID XML Schema attributes. This is done when the change improves clarity in the CoSWID specification. For example, the "name" and "version" SWID fields corresponds to the "software-name" and "software-version" CoSWID fields, respectively. As such, it is not always possible to mechanically translate between corresponding attribute names in the two formats. In such cases, a manual mapping will need to be used. XPath expressions [W3C.REC-xpath20-20101214] need to use SWID names, see Section 5.2.

The 57 human-readable text labels of the CDDL-based CoSWID vocabulary are mapped to integer indices via a block of rules at the bottom of the definition. This allows a more concise integer-based form to be stored or transported, as compared to the less efficient text-based form of the original vocabulary.

Through use of CDDL-based integer labels, CoSWID allows for future expansion in subsequent revisions of this specification and through extensions (see Section 2.2). New constructs can be associated with a new integer index. A deprecated construct can be replaced by a new construct with a new integer index. An implementation can use these integer indexes to identify the construct to parse. The CoSWID Items registry, defined in Section 6.1, is used to ensure that new constructs are assigned a unique index value. This approach avoids the need to have an explicit CoSWID version.

In a number of places, the value encoding admits both integer values and text strings. The integer values are defined in a registry specific to the kind of value; the text values are not intended for interchange and exclusively meant for private use as defined in Section 6.2.2. Encoders SHOULD NOT use string values based on the names registered in the registry, as these values are less concise than their index value equivalent; a decoder MUST however be prepared to accept text strings that are not specified in this document (and ignore the construct if that string is unknown). In the rest of the document, we call this an "integer label with text escape".

The root of the CDDL specification provided by this document is the rule `coswid` (as defined in Section 8):

```
start = coswid
```

In CBOR, an array is encoded using bytes that identify the array, and the array's length or stop point (see [RFC8949]). To make items that support 1 or more values, the following CDDL notation is used.

```
_name_ = (_label_ => _data_ / [ 2* _data_ ])
```

The CDDL rule above allows either a single data item or an array of 2 or more data values to be provided. When a singleton data value is provided, the CBOR markers for the array, array length, and stop point are not needed, saving bytes. When two or more data values are provided, these values are encoded as an array. This modeling pattern is used frequently in the CoSWID CDDL specification to allow for more efficient encoding of singleton values.

Usage of this construct can be simplified using

```
one-or-more<T> = T / [ 2* T ]
```

simplifying the above example to

```
_name_ = (_label_ => one-or-more<_data_>)
```

The following subsections describe the different parts of the CoSWID model.

2.1. Character Encoding

The CDDL "text" type is represented in CBOR as a major type 3, which represents "a string of Unicode characters that [are] encoded as UTF-8 [RFC3629]" (see Section 3.1 of [RFC8949]). Thus both SWID and CoSWID use UTF-8 for the encoding of characters in text strings.

To ensure that UTF-8 character strings are able to be encoded/decoded and exchanged interoperably, text strings in CoSWID MUST be encoded consistent with the Net-Unicode definition defined in [RFC5198].

All names registered with IANA according to requirements in Section 6.2 also MUST be valid according to the XML Schema NMTOKEN data type (see [W3C.REC-xmlschema-2-20041028] Section 3.3.4) to ensure compatibility with the SWID specification where these names are used.

2.2. Concise SWID Extensions

The CoSWID specification contains two features that are not included in the SWID specification on which it is based. These features are:

- * The explicit definition of types for some attributes in the ISO-19770-2:2015 XML representation that are typically represented by the "any attribute" in the SWID model. These are covered in Section 2.4, Paragraph 2.

- * The inclusion of extension points in the CoSWID specification using CDDL sockets (see [RFC8610] Section 3.9). The use of CDDL sockets allow for well-formed extensions to be defined in supplementary CDDL descriptions that support additional uses of CoSWID tags that go beyond the original scope of ISO-19770-2:2015 tags. This extension mechanism can also be used to update the CoSWID format as revisions to ISO-19770-2 are published.

The following CDDL sockets (extension points) are defined in this document, which allow the addition of new information structures to their respective CDDL groups.

Map Name	CDDL Socket	Defined in
concise-swid-tag	\$\$coswid-extension	Section 2.3
entity-entry	\$\$entity-extension	Section 2.6
link-entry	\$\$link-extension	Section 2.7
software-meta-entry	\$\$software-meta-extension	Section 2.8
resource-collection	\$\$resource-collection-extension	Section 2.9.2
file-entry	\$\$file-extension	Section 2.9.2
directory-entry	\$\$directory-extension	Section 2.9.2
process-entry	\$\$process-extension	Section 2.9.2
resource-entry	\$\$resource-extension	Section 2.9.2
payload-entry	\$\$payload-extension	Section 2.9.3
evidence-entry	\$\$evidence-extension	Section 2.9.4

Table 1: CoSWID CDDL Group Extension Points

The CoSWID Items Registry defined in Section 6.1 provides a registration mechanism allowing new items, and their associated index values, to be added to the CoSWID model through the use of the CDDL sockets described in the table above. This registration mechanism provides for well-known index values for data items in CoSWID extensions, allowing these index values to be recognized by implementations supporting a given extension.

The following additional CDDL sockets are defined in this document to allow for adding new values to corresponding type-choices (i.e. to represent enumerations) via custom CDDL specifications.

Enumeration Name	CDDL Socket	Defined in
version-scheme	\$version-scheme	Section 4.1
role	\$role	Section 4.2
ownership	\$ownership	Section 4.3
rel	\$rel	Section 4.4
use	\$use	Section 4.5

Table 2: CoSWID CDDL Enumeration Extension Points

A number of CoSWID value registries are also defined in Section 6.2 that allow new values to be registered with IANA for the enumerations above. This registration mechanism supports the definition of new well-known index values and names for new enumeration values used by CoSWID, which can also be used by other software tagging specifications. This registration mechanism allows new standardized enumerated values to be shared between multiple tagging specifications (and associated implementations) over time.

2.3. The concise-swid-tag Map

The CDDL specification for the root concise-swid-tag map is as follows and this rule and its constraints MUST be followed when creating or validating a CoSWID tag:

```

concise-swid-tag = {
  tag-id => text / bstr .size 16,
  tag-version => integer,
  ? corpus => bool,
  ? patch => bool,
  ? supplemental => bool,
  software-name => text,
  ? software-version => text,
  ? version-scheme => $version-scheme,
  ? media => text,
  ? software-meta => one-or-more<software-meta-entry>,
  entity => one-or-more<entity-entry>,
  ? link => one-or-more<link-entry>,
  ? payload-or-evidence,
  * $$coswid-extension,
  global-attributes,
}

```

```

payload-or-evidence //= ( payload => payload-entry )
payload-or-evidence //= ( evidence => evidence-entry )

```

```

tag-id = 0
software-name = 1
entity = 2
evidence = 3
link = 4
software-meta = 5
payload = 6
corpus = 8
patch = 9
media = 10
supplemental = 11
tag-version = 12
software-version = 13
version-scheme = 14

```

```

$version-scheme /= multipartnumeric
$version-scheme /= multipartnumeric-suffix
$version-scheme /= alphanumeric
$version-scheme /= decimal
$version-scheme /= semver
$version-scheme /= int / text
multipartnumeric = 1
multipartnumeric-suffix = 2
alphanumeric = 3
decimal = 4
semver = 16384

```

The following describes each member of the concise-swid-tag root map.

- * **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 2.4, Paragraph 2.
- * **tag-id (index 0):** A 16-byte binary string, or a textual identifier, uniquely referencing a software component. The tag identifier **MUST** be globally unique. Failure to ensure global uniqueness can create ambiguity in tag use since the tag-id serves as the global key for matching and lookups. If represented as a 16-byte binary string, the identifier **MUST** be a valid universally unique identifier as defined by [RFC4122]. There are no strict guidelines on how the identifier is structured, but examples include a 16-byte GUID (e.g., class 4 UUID) [RFC4122], or a DNS domain name followed by a "/" and a text string, where the domain name serves to ensure uniqueness across organizations. A textual tag-id **MUST NOT** contain a sequence of two underscores ("__", see Section 6.7).
- * **tag-version (index 12):** An integer value that indicate the specific release revision of the tag. Typically, the initial value of this field is set to 0 and the value is increased for subsequent tags produced for the same software component release. This value allows a CoSWID tag producer to correct an incorrect tag previously released without indicating a change to the underlying software component the tag represents. For example, the tag version could be changed to add new metadata, to correct a broken link, to add a missing payload entry, etc. When producing a revised tag, the new tag-version value **MUST** be greater than the old tag-version value.
- * **corpus (index 8):** A boolean value that indicates if the tag identifies and describes an installable software component in its pre-installation state. Installable software includes an 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".

- * `patch` (index 9): A boolean value that indicates if the tag identifies and describes an installed patch that has made incremental changes to a software component installed on an endpoint. If a CoSWID tag is for a patch, the patch item MUST be set to "true". If not provided, the default value MUST be considered "false". A patch item's value MUST NOT be set to "true" if the installation of the associated software package changes the version of a software component.
- * `supplemental` (index 11): A boolean value that indicates if the tag is providing additional information to be associated with another referenced SWID or CoSWID tag. This allows tools and users to record their own metadata about a software component without modifying SWID primary or patch tags created by a software provider. If a CoSWID tag is a supplemental tag, the supplemental item MUST be set to "true". If not provided, the default value MUST be considered "false".
- * `software-name` (index 1): This textual item provides the software component's name. This name is likely the same name that would appear in a package management tool. This item maps to `'/SoftwareIdentity/@name'` in [SWID].
- * `software-version` (index 13): A textual value representing the specific release or development version of the software component. This item maps to `'/SoftwareIdentity/@version'` in [SWID].
- * `version-scheme` (index 14): An integer or textual value representing the versioning scheme used for the software-version item, as an integer label with text escape (Section 2, for the "Version Scheme" registry Section 4.1. . If an integer value is used it MUST be an index value in the range -256 to 65535. Integer values in the range -256 to -1 are reserved for testing and use in closed environments (see Section 6.2.2). Integer values in the range 0 to 65535 correspond to registered entries in the IANA "Software Tag Version Scheme Values" registry (see Section 6.2.4).
- * `media` (index 10): This text value is a hint to the tag consumer to understand what target platform this tag applies to. This item MUST be formatted as a query as defined by the W3C Media Queries Recommendation (see [W3C.REC-css3-mediaqueries-20120619]). Support for media queries are included here for interoperability with [SWID], which does not provide any further requirements for media query use. Thus, this specification does not clarify how a media query is to be used for a CoSWID.

- * `software-meta` (index 5): An open-ended map of key/value data pairs. A number of predefined keys can be used within this item providing for common usage and semantics across the industry. Use of this map allows any additional attribute to be included in the tag. It is expected that industry groups will use a common set of attribute names to allow for interoperability within their communities. Described in Section 2.8. This item maps to `'/SoftwareIdentity/Meta'` in [SWID].
- * `entity` (index 2): Provides information about one or more organizations responsible for producing the CoSWID tag, and producing or releasing the software component referenced by this CoSWID tag. Described in Section 2.6.
- * `link` (index 4): Provides a means to establish relationship arcs between the tag and another items. A given link can be used to establish the relationship between tags or to reference another resource that is related to the CoSWID tag, e.g., vulnerability database association, ROLIE feed [RFC8322], MUD resource [RFC8520], software download location, etc). This is modeled after the HTML "link" element. Described in Section 2.7.
- * `payload` (index 6): This item represents a collection of software artifacts (described by child items) that compose the target software. For example, these artifacts could be the files included with an installer for a corpus tag or installed on an endpoint when the software component is installed for a primary or patch tag. The artifacts listed in a payload may be a superset of the software artifacts that are actually installed. Based on user selections at install time, an installation might not include every artifact that could be created or executed on the endpoint when the software component is installed or run. This item is mutually exclusive to evidence, as payload can only be provided by an external entity. Described in Section 2.9.3.
- * `evidence` (index 3): This item can be used to record the results of a software discovery process used to identify untagged software on an endpoint or to represent indicators for why software is believed to be installed on the endpoint. In either case, a CoSWID tag can be created by the tool performing an analysis of the software components installed on the endpoint. This item is mutually exclusive to payload, as evidence is always generated on the target device ad-hoc. Described in Section 2.9.4.
- * `$$coswid-extension`: This CDDL socket is used to add new information structures to the concise-swid-tag root map. See Section 2.2.

2.4. concise-swid-tag Co-Constraints

The following co-constraints apply to the information provided in the concise-swid-tag group. If any of these constraints is not met, a signed tag cannot be used anymore as a signed statement.

- * The patch and supplemental items MUST NOT both be set to "true".
- * If the patch item is set to "true", the tag SHOULD contain at least one link item (see Section 2.7) with both the rel item value of "patches" and an href item specifying an association with the software that was patched. Without at least one link item the target of the patch cannot be identified and the patch tag cannot be applied without external context.
- * If the supplemental item is set to "true", the tag SHOULD contain at least one link item with both the rel item value of "supplemental" and an href item specifying an association with the software that is supplemented. Without at least one link item the target of supplement tag cannot be identified and the patch tag cannot be applied without external context.
- * 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.

2.5. 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 follows:

```
global-attributes = (  
    ? lang => text,  
    * any-attribute,  
)  
  
any-attribute = (  
    label => one-or-more<text> / one-or-more<int>  
)  
  
label = text / int
```

The following describes each child item of this group.

- * lang (index 15): A textual language tag that conforms with IANA "Language Subtag Registry" [RFC5646]. The context of the specified language applies to all sibling and descendant textual values, unless a descendant object has defined a different language tag. Thus, a new context is established when a descendant object redefines a new language tag. All textual values within a given context MUST be considered expressed in the specified language.
- * any-attribute: This sub-group provides a means to include arbitrary information via label/index ("key") value pairs. Labels can be either a single integer or text string. Values can be a single integer, a text string, or an array of integers or text strings.

2.6. The entity-entry Map

The CDDL for the entity-entry map follows:

```
entity-entry = {  
    entity-name => text,  
    ? reg-id => any-uri,  
    role => one-or-more<$role>,  
    ? thumbprint => hash-entry,  
    * $$entity-extension,  
    global-attributes,  
}
```

```
entity-name = 31  
reg-id = 32  
role = 33  
thumbprint = 34
```

```
$role /= tag-creator  
$role /= software-creator  
$role /= aggregator  
$role /= distributor  
$role /= licensor  
$role /= maintainer  
$role /= int / text  
tag-creator=1  
software-creator=2  
aggregator=3  
distributor=4  
licensor=5  
maintainer=6
```

The following describes each child item of this group.

- * `global-attributes`: The `global-attributes` group described in Section 2.4, Paragraph 2.
- * `entity-name` (index 31): The textual name of the organizational entity claiming the roles specified by the role item for the CoSWID tag. This item maps to `'/SoftwareIdentity/Entity/@name'` in [SWID].
- * `reg-id` (index 32): The registration id value is intended to uniquely identify a naming authority in a given scope (e.g., global, organization, vendor, customer, administrative domain, etc.) for the referenced entity. The value of a registration ID MUST be a RFC 3986 URI; it is not intended to be dereferenced. The scope will usually be the scope of an organization.
- * `role` (index 33): An integer or textual value (integer label with text escape, see Section 2) representing the relationship(s) between the entity, and this tag or the referenced software component. If an integer value is used it MUST be an index value in the range -256 to 255. Integer values in the range -256 to -1 are reserved for testing and use in closed environments (see Section 6.2.2). Integer values in the range 0 to 255 correspond to registered entries in the IANA "Software Tag Entity Role Values" registry (see Section 6.2.5).

The following additional requirements exist for the use of the "role" item:

- An entity item MUST be provided with the role of "tag-creator" for every CoSWID tag. This indicates the organization that created the CoSWID tag.
 - An entity item SHOULD be provided with the role of "software-creator" for every CoSWID tag, if this information is known to the tag creator. This indicates the organization that created the referenced software component.
- * `thumbprint` (index 34): The value of the thumbprint item provides a hash (i.e. the thumbprint) of the signing entity's public key certificate. This provides an indicator of which entity signed the CoSWID tag, which will typically be the tag creator. See Section 2.9.1 for more details on the use of the hash-entry data structure.
 - * `$$entity-extension`: This CDDL socket can be used to extend the entity-entry group model. See Section 2.2.

2.7. The link-entry Map

The CDDL for the link-entry map follows:

```
link-entry = {  
    ? artifact => text,  
    href => any-uri,  
    ? media => text,  
    ? ownership => $ownership,  
    rel => $rel,  
    ? media-type => text,  
    ? use => $use,  
    * $$link-extension,  
    global-attributes,  
}
```

```
media = 10  
artifact = 37  
href = 38  
ownership = 39  
rel = 40  
media-type = 41  
use = 42
```

```
$ownership /= shared  
$ownership /= private  
$ownership /= abandon  
$ownership /= int / text  
abandon=1  
private=2  
shared=3
```

```
$rel /= ancestor  
$rel /= component  
$rel /= feature  
$rel /= installationmedia  
$rel /= packageinstaller  
$rel /= parent  
$rel /= patches  
$rel /= requires  
$rel /= see-also  
$rel /= supersedes  
$rel /= supplemental  
$rel /= -356..65536 / text  
ancestor=1  
component=2  
feature=3  
installationmedia=4
```

```
packageinstaller=5
parent=6
patches=7
requires=8
see-also=9
supersedes=10
supplemental=11

$use /= optional
$use /= required
$use /= recommended
$use /= int / text
optional=1
required=2
recommended=3
```

The following describes each member of this map.

- * `global-attributes`: The `global-attributes` group described in Section 2.4, Paragraph 2.
- * `artifact` (index 37): To be used with `rel="installation-media"`, this item's value provides the absolute filesystem path to the installer executable or script that can be run to launch the referenced installation. Links with the same artifact name MUST be considered mirrors of each other, allowing the installation media to be acquired from any of the described sources.
- * `href` (index 38): A URI-reference [RFC3986] for the referenced resource. The `"href"` item's value can be, but is not limited to, the following (which is a slightly modified excerpt from [SWID]):
 - If no URI scheme is provided, then the URI-reference is a relative reference relative to the base URI of the CoSWID tag, i.e., the URI under which the CoSWID tag was provided. For example, `../folder/supplemental.coswid`.
 - a physical resource location with any acceptable URI scheme (e.g., `file://` `http://` `https://` `ftp://`)
 - a URI with `"swid:"` as the scheme refers to another SWID or CoSWID by the referenced tag's tag-id. This URI needs to be resolved in the context of the endpoint by software that can lookup other SWID or CoSWID tags. For example, `"swid:2df9de35-0aff-4a86-ace6-f7ddddlade4c"` references the tag with the tag-id value `"2df9de35-0aff-4a86-ace6-f7ddddlade4c"`.

- a URI with "swidpath:" as the scheme, which refers to another software tag via an XPATH query [W3C.REC-xpath20-20101214] that matches items in that tag (Section 5.2). This scheme is provided for compatibility with [SWID]. This specification does not define how to resolve an XPATH query in the context of CBOR, see Section 5.2.
- * media (index 10): A hint to the consumer of the link to what target platform the link is applicable to. This item represents a query as defined by the W3C Media Queries Recommendation (see [W3C.REC-css3-mediaqueries-20120619]). As highlighted in media defined in Section 2.3, support for media queries are included here for interoperability with [SWID], which does not provide any further requirements for media query use. Thus, this specification does not clarify how a media query is to be used for a CoSWID.
- * ownership (index 39): An integer or textual value (integer label with text escape, see Section 2, for the "Software Tag Link Ownership Values" registry Section 4.3) used when the "href" item references another software component to indicate the degree of ownership between the software component referenced by the CoSWID tag and the software component referenced by the link. If an integer value is used it MUST be an index value in the range -256 to 255. Integer values in the range -256 to -1 are reserved for testing and use in closed environments (see Section 6.2.2). Integer values in the range 0 to 255 correspond to registered entries in the "Software Tag Link Ownership Values" registry.

- * `rel` (index 40): An integer or textual value that (integer label with text escape, see Section 2, for the "Software Tag Link Relationship Values" registry Section 4.3) identifies the relationship between this CoSWID and the target resource identified by the `href` item. If an integer value is used it MUST be an index value in the range -256 to 65535. Integer values in the range -256 to -1 are reserved for testing and use in closed environments (see Section 6.2.2). Integer values in the range 0 to 65535 correspond to registered entries in the IANA "Software Tag Link Relationship Values" registry (see Section 6.2.7. If a string value is used it MUST be either a private use name as defined in Section 6.2.2 or a "Relation Name" from the IANA "Link Relation Types" registry: <https://www.iana.org/assignments/link-relations/link-relations.xhtml> as defined by [RFC8288]. When a string value defined in the IANA "Software Tag Link Relationship Values" registry matches a Relation Name defined in the IANA "Link Relation Types" registry, the index value in the IANA "Software Tag Link Relationship Values" registry MUST be used instead, as this relationship has a specialized meaning in the context of a CoSWID tag. String values correspond to registered entries in the "Software Tag Link Relationship Values" registry.
- * `media-type` (index 41): A link can point to arbitrary resources on the endpoint, local network, or Internet using the `href` item. Use of this item supplies the resource consumer with a hint of what type of resource to expect. (This is a `_hint_`: There is no obligation for the server hosting the target of the URI to use the indicated media type when the URI is dereferenced.) Media types are identified by referencing a "Name" from the IANA "Media Types" registry: <http://www.iana.org/assignments/media-types/media-types.xhtml>. This item maps to `'/SoftwareIdentity/Link/@type'` in [SWID].
- * `use` (index 42): An integer or textual value (integer label with text escape, see Section 2, for the "Software Tag Link Relationship Values" registry Section 4.3) used to determine if the referenced software component has to be installed before installing the software component identified by the CoSWID tag. If an integer value is used it MUST be an index value in the range -256 to 255. Integer values in the range -256 to -1 are reserved for testing and use in closed environments (see Section 6.2.2). Integer values in the range 0 to 255 correspond to registered entries in the IANA "Link Use Values" registry (see Section 6.2.8. If a string value is used it MUST be a private use name as defined in Section 6.2.2. String values correspond to registered entries in the "Software Tag Link Use Values" registry.

- * `$$link-extension`: This CDDL socket can be used to extend the link-entry map model. See Section 2.2.

2.8. The software-meta-entry Map

The CDDL for the software-meta-entry map follows:

```
software-meta-entry = {  
  ? activation-status => text,  
  ? channel-type => text,  
  ? colloquial-version => text,  
  ? description => text,  
  ? edition => text,  
  ? entitlement-data-required => bool,  
  ? entitlement-key => text,  
  ? generator => text / bstr .size 16,  
  ? persistent-id => text,  
  ? product => text,  
  ? product-family => text,  
  ? revision => text,  
  ? summary => text,  
  ? unspsc-code => text,  
  ? unspsc-version => text,  
  * $$software-meta-extension,  
  global-attributes,  
}
```

```
activation-status = 43  
channel-type = 44  
colloquial-version = 45  
description = 46  
edition = 47  
entitlement-data-required = 48  
entitlement-key = 49  
generator = 50  
persistent-id = 51  
product = 52  
product-family = 53  
revision = 54  
summary = 55  
unspsc-code = 56  
unspsc-version = 57
```

The following describes each child item of this group.

- * `global-attributes`: The global-attributes group described in Section 2.4, Paragraph 2.

- * `activation-status` (index 43): A textual value that identifies how the software component has been activated, which might relate to specific terms and conditions for its use (e.g., Trial, Serialized, Licensed, Unlicensed, etc) and relate to an entitlement. This attribute is typically used in supplemental tags as it contains information that might be selected during a specific install.
- * `channel-type` (index 44): A textual value that identifies which sales, licensing, or marketing channel the software component has been targeted for (e.g., Volume, Retail, OEM, Academic, etc). This attribute is typically used in supplemental tags as it contains information that might be selected during a specific install.
- * `colloquial-version` (index 45): A textual value for the software component's informal or colloquial version. Examples may include a year value, a major version number, or similar value that are used to identify a group of specific software component releases that are part of the same release/support cycle. This version can be the same through multiple releases of a software component, while the software-version specified in the concise-swid-tag group is much more specific and will change for each software component release. This version is intended to be used for string comparison (byte-by-byte) only and is not intended to be used to determine if a specific value is earlier or later in a sequence.
- * `description` (index 46): A textual value that provides a detailed description of the software component. This value MAY be multiple paragraphs separated by CR LF characters as described by [RFC5198].
- * `edition` (index 47): A textual value indicating that the software component represents a functional variation of the code base used to support multiple software components. For example, this item can be used to differentiate enterprise, standard, or professional variants of a software component.
- * `entitlement-data-required` (index 48): A boolean value that can be used to determine if accompanying proof of entitlement is needed when a software license reconciliation process is performed.
- * `entitlement-key` (index 49): A vendor-specific textual key that can be used to identify and establish a relationship to an entitlement. Examples of an entitlement-key might include a serial number, product key, or license key. For values that relate to a given software component install (i.e., license key), a supplemental tag will typically contain this information. In

other cases, where a general-purpose key can be provided that applies to all possible installs of the software component on different endpoints, a primary tag will typically contain this information. Since CoSWID tags are not intended to contain confidential information, tag authors are advised not to record unprotected, private software license keys in this field.

- * generator (index 50): The name (or tag-id) of the software component that created the CoSWID tag. If the generating software component has a SWID or CoSWID tag, then the tag-id for the generating software component SHOULD be provided.
- * persistent-id (index 51): A globally unique identifier used to identify a set of software components that are related. Software components sharing the same persistent-id can be different versions. This item can be used to relate software components, released at different points in time or through different release channels, that may not be able to be related through use of the link item.
- * product (index 52): A basic name for the software component that can be common across multiple tagged software components (e.g., Apache HTTPD).
- * product-family (index 53): A textual value indicating the software components overall product family. This should be used when multiple related software components form a larger capability that is installed on multiple different endpoints. For example, some software families may consist of server, client, and shared service components that are part of a larger capability. Email systems, enterprise applications, backup services, web conferencing, and similar capabilities are examples of families. Use of this item is not intended to represent groups of software that are bundled or installed together. The persistent-id or link items SHOULD be used to relate bundled software components.
- * revision (index 54): A string value indicating an informal or colloquial release version of the software. This value can provide a different version value as compared to the software-version specified in the concise-swid-tag group. This is useful when one or more releases need to have an informal version label that differs from the specific exact version value specified by software-version. Examples can include SP1, RC1, Beta, etc.
- * summary (index 55): A short description of the software component. This MUST be a single sentence suitable for display in a user interface.

- * `unspsc-code` (index 56): An 8 digit UNSPSC classification code for the software component as defined by the United Nations Standard Products and Services Code (UNSPSC, [UNSPSC]).
- * `unspsc-version` (index 57): The version of UNSPSC used to define the `unspsc-code` value.
- * `$$meta-extension`: This CDDL socket can be used to extend the software-meta-entry group model. See Section 2.2.

2.9. The Resource Collection Definition

2.9.1. The hash-entry Array

CoSWID adds explicit support for the representation of hash entries using algorithms that are registered in the IANA "Named Information Hash Algorithm Registry" [IANA.named-information] using the hash member (index 7) and the corresponding hash-entry type. This is the equivalent of the namespace qualified "hash" attribute in [SWID].

```
hash-entry = [  
    hash-alg-id: int,  
    hash-value: bytes,  
]
```

The number used as a value for `hash-alg-id` is an integer-based hash algorithm identifier whose value MUST refer to an ID in the IANA "Named Information Hash Algorithm Registry" [IANA.named-information] with a Status of "current" (at the time the generator software was built or later); other hash algorithms MUST NOT be used. If the `hash-alg-id` is not known, then the integer value "0" MUST be used. This allows for conversion from ISO SWID tags [SWID], which do not allow an algorithm to be identified for this field.

The `hash-value` MUST represent the raw hash value as a byte string (as opposed to, e.g., base64 encoded) generated from the representation of the resource using the hash algorithm indicated by `hash-alg-id`.

2.9.2. The resource-collection Group

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

The CDDL for the resource-collection group follows:

```
path-elements-group = ( ? directory => one-or-more<directory-entry>,  
                        ? file => one-or-more<file-entry>,  
                        )  
  
resource-collection = (  
    path-elements-group,  
    ? process => one-or-more<process-entry>,  
    ? resource => one-or-more<resource-entry>,  
    * $$resource-collection-extension,  
)  
  
filesystem-item = (  
    ? key => bool,  
    ? location => text,  
    fs-name => text,  
    ? root => text,  
)  
  
file-entry = {  
    filesystem-item,  
    ? size => uint,  
    ? file-version => text,  
    ? hash => hash-entry,  
    * $$file-extension,  
    global-attributes,  
}  
  
directory-entry = {  
    filesystem-item,  
    ? path-elements => { path-elements-group },  
    * $$directory-extension,  
    global-attributes,  
}  
  
process-entry = {  
    process-name => text,  
    ? pid => integer,  
    * $$process-extension,  
    global-attributes,  
}  
  
resource-entry = {  
    type => text,  
    * $$resource-extension,  
    global-attributes,  
}  
  
directory = 16
```

file = 17
process = 18
resource = 19
size = 20
file-version = 21
key = 22
location = 23
fs-name = 24
root = 25
path-elements = 26
process-name = 27
pid = 28
type = 29

The following describes each member of the groups and maps illustrated above.

- * filesystem-item: A list of common items used for representing the filesystem root, relative location, name, and significance of a file or directory item.
- * global-attributes: The global-attributes group described in Section 2.4, Paragraph 2.
- * directory (index 16): A directory item allows child directory and file items to be defined within a directory hierarchy for the software component.
- * file (index 17): A file item allows details about a file to be provided for the software component.
- * process (index 18): A process item allows details to be provided about the runtime behavior of the software component, such as information that will appear in a process listing on an endpoint.
- * resource (index 19): A resource item can be used to provide details about an artifact or capability expected to be found on an endpoint or evidence collected related to the software component. This can be used to represent concepts not addressed directly by the directory, file, or process items. Examples include: registry keys, bound ports, etc. The equivalent construct in [SWID] is currently under specified. As a result, this item might be further defined through extension in the future.
- * size (index 20): The file's size in bytes.

- * file-version (index 21): The file's version as reported by querying information on the file from the operating system (if available). This item maps to `'/SoftwareIdentity/(Payload|Evidence)/File/@version'` in [SWID].
- * hash (index 7): A hash of the file as described in Section 2.9.1.
- * key (index 22): A boolean value indicating if a file or directory is significant or required for the software component to execute or function properly. These are files or directories that can be used to affirmatively determine if the software component is installed on an endpoint.
- * location (index 23): The filesystem path where a file is expected to be located when installed or copied. The location MUST be either relative to the location of the parent directory item (preferred), or relative to the location of the CoSWID tag (as indicated in the location value in the evidence entry map) if no parent is defined. The location MUST NOT include a file's name, which is provided by the fs-name item.
- * fs-name (index 24): The name of the directory or file without any path information. This aligns with a file "name" in [SWID]. This item maps to `'/SoftwareIdentity/(Payload|Evidence)/(File|Directory)/@name'` in [SWID].
- * root (index 25): A host-specific name for the root of the filesystem. The location item is considered relative to this location if specified. If not provided, the value provided by the location item is expected to be relative to its parent or the location of the CoSWID tag if no parent is provided.
- * path-elements (index 26): This group allows a hierarchy of directory and file items to be defined in payload or evidence items. This is a construction within the CDDL definition of CoSWID to support shared syntax and does not appear in [SWID].
- * process-name (index 27): The software component's process name as it will appear in an endpoint's process list. This aligns with a process "name" in [SWID]. This item maps to `'/SoftwareIdentity/(Payload|Evidence)/Process/@name'` in [SWID].
- * pid (index 28): The process ID identified for a running instance of the software component in the endpoint's process list. This is used as part of the evidence item.

- * `type (index 29)`: A human-readable string indicating the type of resource.
- * `$$resource-collection-extension`: This CDDL socket can be used to extend the resource-collection group model. This can be used to add new specialized types of resources. See Section 2.2.
- * `$$file-extension`: This CDDL socket can be used to extend the file-entry group model. See Section 2.2.
- * `$$directory-extension`: This CDDL socket can be used to extend the directory-entry group model. See Section 2.2.
- * `$$process-extension`: This CDDL socket can be used to extend the process-entry group model. See Section 2.2.
- * `$$resource-extension`: This CDDL socket can be used to extend the resource-entry group model. See Section 2.2.

2.9.3. The payload-entry Map

The CDDL for the payload-entry map follows:

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

The following describes each child item of this group.

- * `global-attributes`: The global-attributes group described in Section 2.4, Paragraph 2.
- * `resource-collection`: The resource-collection group described in Section 2.9.2.
- * `$$payload-extension`: This CDDL socket can be used to extend the payload-entry group model. See Section 2.2.

2.9.4. The evidence-entry Map

The CDDL for the evidence-entry map follows:

```
evidence-entry = {  
  resource-collection,  
  ? date => integer-time,  
  ? device-id => text,  
  ? location => text,  
  * $$evidence-extension,  
  global-attributes,  
}
```

```
date = 35  
device-id = 36
```

The following describes each child item of this group.

- * **global-attributes**: The global-attributes group described in Section 2.4, Paragraph 2.
- * **resource-collection**: The resource-collection group described in Section 2.9.2.
- * **date (index 35)**: The date and time the information was collected pertaining to the evidence item.
- * **device-id (index 36)**: The endpoint's string identifier from which the evidence was collected.
- * **location (index 23)**: The absolute filepath of the location of the CoSWID tag generated as evidence. (Location values in filesystem-items in the payload can be expressed relative to this location.)
- * **\$\$evidence-extension**: This CDDL socket can be used to extend the evidence-entry group model. See Section 2.2.

2.10. Full CDDL Specification

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

```
<CODE BEGINS>  
concise-swid-tag = {  
  tag-id => text / bstr .size 16,  
  tag-version => integer,  
  ? corpus => bool,  
  ? patch => bool,  
  ? supplemental => bool,  
  software-name => text,  
  ? software-version => text,
```

```
? version-scheme => $version-scheme,
? media => text,
? software-meta => one-or-more<software-meta-entry>,
entity => one-or-more<entity-entry>,
? link => one-or-more<link-entry>,
? payload-or-evidence,
* $$coswid-extension,
global-attributes,
}

payload-or-evidence //= ( payload => payload-entry )
payload-or-evidence //= ( evidence => evidence-entry )

any-uri = uri
label = text / int

$version-scheme /= multipartnumeric
$version-scheme /= multipartnumeric-suffix
$version-scheme /= alphanumeric
$version-scheme /= decimal
$version-scheme /= semver
$version-scheme /= int / text

any-attribute = (
  label => one-or-more<text> / one-or-more<int>
)

one-or-more<T> = T / [ 2* T ]

global-attributes = (
  ? lang => text,
  * any-attribute,
)

hash-entry = [
  hash-alg-id: int,
  hash-value: bytes,
]

entity-entry = {
  entity-name => text,
  ? reg-id => any-uri,
  role => one-or-more<$role>,
  ? thumbprint => hash-entry,
  * $$entity-extension,
  global-attributes,
}
```

```
$role /= tag-creator
$role /= software-creator
$role /= aggregator
$role /= distributor
$role /= licenser
$role /= maintainer
$role /= int / text

link-entry = {
  ? artifact => text,
  href => any-uri,
  ? media => text,
  ? ownership => $ownership,
  rel => $rel,
  ? media-type => text,
  ? use => $use,
  * $$link-extension,
  global-attributes,
}

$ownership /= shared
$ownership /= private
$ownership /= abandon
$ownership /= int / text

$rel /= ancestor
$rel /= component
$rel /= feature
$rel /= installationmedia
$rel /= packageinstaller
$rel /= parent
$rel /= patches
$rel /= requires
$rel /= see-also
$rel /= supersedes
$rel /= supplemental
$rel /= -256..64436 / text

$use /= optional
$use /= required
$use /= recommended
$use /= int / text

software-meta-entry = {
  ? activation-status => text,
  ? channel-type => text,
  ? colloquial-version => text,
  ? description => text,
```

```
? edition => text,
? entitlement-data-required => bool,
? entitlement-key => text,
? generator => text / bstr .size 16,
? persistent-id => text,
? product => text,
? product-family => text,
? revision => text,
? summary => text,
? unspsc-code => text,
? unspsc-version => text,
* $$software-meta-extension,
global-attributes,
}

path-elements-group = ( ? directory => one-or-more<directory-entry>,
                        ? file => one-or-more<file-entry>,
                        )

resource-collection = (
  path-elements-group,
  ? process => one-or-more<process-entry>,
  ? resource => one-or-more<resource-entry>,
  * $$resource-collection-extension,
)

file-entry = {
  filesystem-item,
  ? size => uint,
  ? file-version => text,
  ? hash => hash-entry,
  * $$file-extension,
  global-attributes,
}

directory-entry = {
  filesystem-item,
  ? path-elements => { path-elements-group },
  * $$directory-extension,
  global-attributes,
}

process-entry = {
  process-name => text,
  ? pid => integer,
  * $$process-extension,
  global-attributes,
}
```

```
resource-entry = {
  type => text,
  * $$resource-extension,
  global-attributes,
}

filesystem-item = (
  ? key => bool,
  ? location => text,
  fs-name => text,
  ? root => text,
)

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

evidence-entry = {
  resource-collection,
  ? date => integer-time,
  ? device-id => text,
  ? location => text,
  * $$evidence-extension,
  global-attributes,
}

integer-time = #6.1(int)

; "global map member" integer indexes
tag-id = 0
software-name = 1
entity = 2
evidence = 3
link = 4
software-meta = 5
payload = 6
hash = 7
corpus = 8
patch = 9
media = 10
supplemental = 11
tag-version = 12
software-version = 13
version-scheme = 14
lang = 15
directory = 16
```

```
file = 17
process = 18
resource = 19
size = 20
file-version = 21
key = 22
location = 23
fs-name = 24
root = 25
path-elements = 26
process-name = 27
pid = 28
type = 29
entity-name = 31
reg-id = 32
role = 33
thumbprint = 34
date = 35
device-id = 36
artifact = 37
href = 38
ownership = 39
rel = 40
media-type = 41
use = 42
activation-status = 43
channel-type = 44
colloquial-version = 45
description = 46
edition = 47
entitlement-data-required = 48
entitlement-key = 49
generator = 50
persistent-id = 51
product = 52
product-family = 53
revision = 54
summary = 55
unspsc-code = 56
unspsc-version = 57

; "version-scheme" integer indexes
multipartnumeric = 1
multipartnumeric-suffix = 2
alphanumeric = 3
decimal = 4
semver = 16384
```

```
; "role" integer indexes
tag-creator=1
software-creator=2
aggregator=3
distributor=4
licensor=5
maintainer=6

; "ownership" integer indexes
abandon=1
private=2
shared=3

; "rel" integer indexes
ancestor=1
component=2
feature=3
installationmedia=4
packageinstaller=5
parent=6
patches=7
requires=8
see-also=9
supersedes=10
; supplemental=11 ; this is already defined earlier

; "use" integer indexes
optional=1
required=2
recommended=3
<CODE ENDS>
```

3. Determining the Type of CoSWID

The operational model for SWID and CoSWID tags was introduced in Section 1.1, which described four different CoSWID tag types. The following additional rules apply to the use of CoSWID tags to ensure that created tags properly identify the tag type.

The first matching rule MUST determine the type of the CoSWID tag.

1. Primary Tag: A CoSWID tag MUST be considered a primary tag if the corpus, patch, and supplemental items are "false".
2. Supplemental Tag: A CoSWID tag MUST be considered a supplemental tag if the supplemental item is set to "true".

3. Corpus Tag: A CoSWID tag MUST be considered a corpus tag if the corpus item is "true".
4. Patch Tag: A CoSWID tag MUST be considered a patch tag if the patch item is "true".

Note: Multiple of the corpus, patch, and supplemental items can have values set as "true". The rules above provide a means to determine the tag's type in such a case. For example, a SWID or CoSWID tag for a patch installer might have both corpus and patch items set to "true". In such a case, the tag is a "Corpus Tag". The tag installed by this installer would have only the patch item set to "true", making the installed tag type a "Patch Tag".

4. CoSWID Indexed Label Values

This section defines a number of kinds of indexed label values that are maintained in a registry each (Section 6). These values are represented as positive integers. In each registry, the value 0 is marked as Reserved.

4.1. Version Scheme

The following table contains a set of values for use in the concise-swid-tag group's version-scheme item. Version Scheme Name strings match the version schemes defined in the ISO/IEC 19770-2:2015 [SWID] specification. Index value indicates the value to use as the version-scheme item's value. The Version Scheme Name provides human-readable text for the value. The Definition describes the syntax of allowed values for each entry.

Index	Version Scheme Name	Definition
1	multipartnumeric	Numbers separated by dots, where the numbers are interpreted as decimal integers (e.g., 1.2.3, 1.2.3.4.5.6.7, 1.4.5, 1.21)
2	multipartnumeric+suffix	Numbers separated by dots, where the numbers are interpreted as decimal integers with an additional textual suffix (e.g., 1.2.3a)
3	alphanumeric	Strictly a string, no interpretation as number
4	decimal	A single decimal floating point number
16384	semver	A semantic version as defined by [SWID]. Also see the [SEMVER] specification for more information

Table 3: Version Scheme Values

multipartnumeric and the numbers part of multipartnumeric+suffix are interpreted as a sequence of numbers and are sorted in lexicographical order by these numbers (i.e., not by the digits in the numbers) and then the textual suffix (for multipartnumeric+suffix). Alphanumeric strings are sorted lexicographically as character strings. Decimal version numbers are interpreted as a single floating point number (e.g., 1.25 is less than 1.3).

The values above are registered in the IANA "Software Tag Version Scheme Values" registry defined in Section 6.2.4. Additional entries will likely be registered over time in this registry.

A CoSWID producer that is aware of the version scheme that has been used to select the version value, SHOULD include the optional version-scheme item to avoid semantic ambiguity. If the CoSWID producer does not have this information, it SHOULD omit the version-scheme item. The following heuristics can be used by a CoSWID consumer, based on the version schemes' partially overlapping value spaces:

- * "decimal" and "multipartnumeric" partially overlap in their value space when a value matches a decimal number. When a corresponding software-version item's value falls within this overlapping value space, the "decimal" version scheme SHOULD be assumed.
- * "multipartnumeric" and "semver" partially overlap in their value space when a "multipartnumeric" value matches the semantic versioning syntax. When a corresponding software-version item's value falls within this overlapping value space, the "semver" version scheme SHOULD be assumed.
- * "alphanumeric" and other version schemes might overlap in their value space. When a corresponding software-version item's value falls within this overlapping value space, the other version scheme SHOULD be assumed instead of "alphanumeric".

Note that these heuristics are imperfect and can guess wrong, which is the reason the version-scheme item SHOULD be included by the producer.

4.2. Entity Role Values

The following table indicates the index value to use for the entity-entry group's role item (see Section 2.6). These values match the entity roles defined in the ISO/IEC 19770-2:2015 [SWID] specification. The "Index" value indicates the value to use as the role item's value. The "Role Name" provides human-readable text for the value. The "Definition" describes the semantic meaning of each entry.

Index	Role Name	Definition
1	tagCreator	The person or organization that created the containing SWID or CoSWID tag
2	softwareCreator	The person or organization entity that created the software component.
3	aggregator	From [SWID], "An organization or system that encapsulates software from their own and/or other organizations into a different distribution process (as in the case of virtualization), or as a completed system to accomplish a specific task (as in the case of a value added reseller)."
4	distributor	From [SWID], "An entity that furthers the marketing, selling and/or distribution of software from the original place of manufacture to the ultimate user without modifying the software, its packaging or its labelling."
5	licensor	From [SAM] as "software licensor", a "person or organization who owns or holds the rights to issue a software license for a specific software [component]"
6	maintainer	The person or organization that is responsible for coordinating and making updates to the source code for the software component. This SHOULD be used when the "maintainer" is a different person or organization than the original "softwareCreator".

Table 4: Entity Role Values

The values above are registered in the IANA "Software Tag Entity Role Values" registry defined in Section 6.2.5. Additional values will likely be registered over time.

4.3. Link Ownership Values

The following table indicates the index value to use for the link-entry group's ownership item (see Section 2.7). These values match the link ownership values defined in the ISO/IEC 19770-2:2015 [SWID] specification. The "Index" value indicates the value to use as the link-entry group ownership item's value. The "Ownership Type" provides human-readable text for the value. The "Definition" describes the semantic meaning of each entry.

Index	Ownership Type	Definition
1	abandon	If the software component referenced by the CoSWID tag is uninstalled, then the referenced software SHOULD NOT be uninstalled
2	private	If the software component referenced by the CoSWID tag is uninstalled, then the referenced software SHOULD be uninstalled as well.
3	shared	If the software component referenced by the CoSWID tag is uninstalled, then the referenced software SHOULD be uninstalled if no other components sharing the software.

Table 5: Link Ownership Values

The values above are registered in the IANA "Software Tag Link Ownership Values" registry defined in Section 6.2.6. Additional values will likely be registered over time.

4.4. Link Rel Values

The following table indicates the index value to use for the link-entry group's rel item (see Section 2.7). These values match the link rel values defined in the ISO/IEC 19770-2:2015 [SWID] specification. The "Index" value indicates the value to use as the link-entry group ownership item's value. The "Relationship Type" provides human-readable text for the value. The "Definition" describes the semantic meaning of each entry.

Index	Relationship Type	Definition
1	ancestor	The link references a software tag for a previous release of this software. This can be useful to define an upgrade path.
2	component	The link references a software tag for a separate component of this software.
3	feature	The link references a configurable feature of this software that can be enabled or disabled without changing the installed files.
4	installationmedia	The link references the installation package that can be used to install this software.
5	packageinstaller	The link references the installation software needed to install this software.
6	parent	The link references a software tag that is the parent of the referencing tag. This relationship can be used when multiple software components are part of a software bundle, where the "parent" is the software tag for the bundle, and each child is a "component". In such a case, each child component can provide a "parent" link relationship to the bundle's software tag, and the bundle can provide a "component" link relationship to each child software component.
7	patches	The link references a software tag that the referencing software patches. Typically only used for patch tags (see Section 1.1).
8	requires	The link references a

		prerequisite for installing this software. A patch tag (see Section 1.1) can use this to represent base software or another patch that needs to be installed first.
9	see-also	The link references other software that may be of interest that relates to this software.
10	supersedes	The link references another software that this software replaces. A patch tag (see Section 1.1) can use this to represent another patch that this patch incorporates or replaces.
11	supplemental	The link references a software tag that the referencing tag supplements. Used on supplemental tags (see Section 1.1).

Table 6: Link Relationship Values

The values above are registered in the IANA "Software Tag Link Relationship Values" registry defined in Section 6.2.7. Additional values will likely be registered over time.

4.5. Link Use Values

The following table indicates the index value to use for the link-entry group's use item (see Section 2.7). These values match the link use values defined in the ISO/IEC 19770-2:2015 [SWID] specification. The "Index" value indicates the value to use as the link-entry group use item's value. The "Use Type" provides human-readable text for the value. The "Definition" describes the semantic meaning of each entry.

Index	Use Type	Definition
1	optional	From [SWID], "Not absolutely required; the [Link]'d software is installed only when specified."
2	required	From [SWID], "The [Link]'d software is absolutely required for an operation software installation."
3	recommended	From [SWID], "Not absolutely required; the [Link]'d software is installed unless specified otherwise."

Table 7: Link Use Values

The values above are registered in the IANA "Software Tag Link Use Values" registry defined in Section 6.2.8. Additional values will likely be registered over time.

5. URI Schemes

This specification defines the following URI schemes for use in CoSWID and to provide interoperability with schemes used in [SWID].

Note: These URI schemes are used in [SWID] without an IANA registration. The present specification ensures that these URI schemes are properly defined going forward.

// RFC Ed.: throughout this section, please replace RFC-AAAA with the
// RFC number of this specification and remove this note.

5.1. "swid" URI Scheme

There is a need for a scheme name that can be used in URIs that point to a specific software tag by that tag's tag-id, such as the use of the link entry as described in Section 2.7. Since this scheme is used both in a standards track document and an ISO standard, this scheme needs to be used without fear of conflicts with current or future actual schemes. In Section 6.6.1, the scheme "swid" is registered as a 'permanent' scheme for that purpose.

URIs specifying the "swid" scheme are used to reference a software tag by its tag-id. A tag-id referenced in this way can be used to identify the tag resource in the context of where it is referenced

from. For example, when a tag is installed on a given device, that tag can reference related tags on the same device using URIs with this scheme.

For URIs that use the "swid" scheme, the scheme specific part MUST consist of a referenced software tag's tag-id. This tag-id MUST be URI encoded according to [RFC3986] Section 2.1.

The following expression is a valid example:

```
swid:2df9de35-0aff-4a86-ace6-f7dddlade4c
```

5.2. "swidpath" URI Scheme

There is a need for a scheme name that can be used in URIs to identify a collection of specific software tags with data elements that match an XPath expression, such as the use of the link entry as described in Section 2.7. The scheme named "swidpath" is used for this purpose in [SWID], but not registered. To enable usage without fear of conflicts with current or future actual schemes, the present document registers it as a 'permanent' scheme for that purpose (see Section 6.6.2).

URIs specifying the "swidpath" scheme are used to filter tags out of a base collection, so that matching tags are included in the identified tag collection. The XPath expression [W3C.REC-xpath20-20101214] references the data that must be found in a given software tag out of base collection for that tag to be considered a matching tag. Tags to be evaluated (the base collection) include all tags in the context of where the "swidpath URI" is referenced from. For example, when a tag is installed on a given device, that tag can reference related tags on the same device using a URI with this scheme.

For URIs that use the "swidpath" scheme, the following requirements apply:

- * The scheme specific part MUST be an XPath expression as defined by [W3C.REC-xpath20-20101214]. The included XPath expression will be URI encoded according to [RFC3986] Section 2.1.
- * This XPath is evaluated over SWID tags, or COSWID tags transformed into SWID tags, found on a system. A given tag MUST be considered a match if the XPath evaluation result value has an effective boolean value of "true" according to [W3C.REC-xpath20-20101214] Section 2.4.3.

6. IANA Considerations

This document has a number of IANA considerations, as described in the following subsections. In summary, 6 new registries are established with this request, with initial entries provided for each registry. New values for 5 other registries are also requested.

6.1. CoSWID Items Registry

This registry uses integer values as index values in CBOR maps.

This document defines a new registry titled "CoSWID Items". Future registrations for this registry are to be made based on [BCP26] as follows:

Range	Registration Procedures
0-32767	Standards Action with Expert Review
32768-4294967295	Specification Required

Table 8: CoSWID Items Registration Procedures

All negative values are reserved for Private Use.

Initial registrations for the "CoSWID Items" registry are provided below. Assignments consist of an integer index value, the item name, and a reference to the defining specification.

Index	Item Name	Specification
0	tag-id	RFC-AAAA
1	software-name	RFC-AAAA
2	entity	RFC-AAAA
3	evidence	RFC-AAAA
4	link	RFC-AAAA
5	software-meta	RFC-AAAA
6	payload	RFC-AAAA

7	hash	RFC-AAAA
8	corpus	RFC-AAAA
9	patch	RFC-AAAA
10	media	RFC-AAAA
11	supplemental	RFC-AAAA
12	tag-version	RFC-AAAA
13	software-version	RFC-AAAA
14	version-scheme	RFC-AAAA
15	lang	RFC-AAAA
16	directory	RFC-AAAA
17	file	RFC-AAAA
18	process	RFC-AAAA
19	resource	RFC-AAAA
20	size	RFC-AAAA
21	file-version	RFC-AAAA
22	key	RFC-AAAA
23	location	RFC-AAAA
24	fs-name	RFC-AAAA
25	root	RFC-AAAA
26	path-elements	RFC-AAAA
27	process-name	RFC-AAAA
28	pid	RFC-AAAA
29	type	RFC-AAAA
30	Unassigned	

31	entity-name	RFC-AAAA
32	reg-id	RFC-AAAA
33	role	RFC-AAAA
34	thumbprint	RFC-AAAA
35	date	RFC-AAAA
36	device-id	RFC-AAAA
37	artifact	RFC-AAAA
38	href	RFC-AAAA
39	ownership	RFC-AAAA
40	rel	RFC-AAAA
41	media-type	RFC-AAAA
42	use	RFC-AAAA
43	activation-status	RFC-AAAA
44	channel-type	RFC-AAAA
45	colloquial-version	RFC-AAAA
46	description	RFC-AAAA
47	edition	RFC-AAAA
48	entitlement-data-required	RFC-AAAA
49	entitlement-key	RFC-AAAA
50	generator	RFC-AAAA
51	persistent-id	RFC-AAAA
52	product	RFC-AAAA
53	product-family	RFC-AAAA
54	revision	RFC-AAAA

55	summary	RFC-AAAA	
+-----+	+-----+	+-----+	+-----+
56	unspsc-code	RFC-AAAA	
+-----+	+-----+	+-----+	+-----+
57	unspsc-version	RFC-AAAA	
+-----+	+-----+	+-----+	+-----+
58-4294967295	Unassigned		
+-----+	+-----+	+-----+	+-----+

Table 9: CoSWID Items Initial Registrations

6.2. Software Tag Values Registries

The following IANA registries provide a mechanism for new values to be added over time to common enumerations used by SWID and CoSWID. While neither the CoSWID nor SWID specification is subordinate to the other and will evolve as their respective standards group chooses, there is value in supporting alignment between the two standards. Shared use of common code points, as spelled out in these registries, will facilitate this alignment, hence the intent for shared use of these registries and the decision to use "swid" (rather than "coswid") in registry names.

6.2.1. Registration Procedures

The following registries allow for the registration of index values and names. New registrations will be permitted through either a Standards Action with Expert Review policy or a Specification Required policy [BCP26].

The following registries also reserve the integer-based index values in the range of -1 to -256 for private use as defined by [BCP26] in Section 4.1. This allows values -1 to -24 to be expressed as a single uint_8t in CBOR, and values -25 to -256 to be expressed using an additional uint_8t in CBOR.

6.2.2. Private Use of Index and Name Values

The integer-based index values in the private use range (-1 to -256) are intended for testing purposes and closed environments; values in other ranges SHOULD NOT be assigned for testing.

For names that correspond to private use index values, an Internationalized Domain Name prefix MUST be used to prevent name conflicts using the form:

domainprefix/name

Where both "domainprefix" and "name" MUST each be either an NR-LDH label or a U-label as defined by [RFC5890], and "name" also MUST be a unique name within the namespace defined by the "domainprefix". Use of a prefix in this way allows for a name to be used in the private use range. This is consistent with the guidance in [BCP178].

6.2.3. Expert Review Criteria

Designated experts MUST ensure that new registration requests meet the following additional criteria:

- * The requesting specification MUST provide a clear semantic definition for the new entry. This definition MUST clearly differentiate the requested entry from other previously registered entries.
- * The requesting specification MUST describe the intended use of the entry, including any co-constraints that exist between the use of the entry's index value or name, and other values defined within the SWID/CoSWID model.
- * Index values and names outside the private use space MUST NOT be used without registration. This is considered squatting and MUST be avoided. Designated experts MUST ensure that reviewed specifications register all appropriate index values and names.
- * Standards track documents MAY include entries registered in the range reserved for entries under the Specification Required policy. This can occur when a standards track document provides further guidance on the use of index values and names that are in common use, but were not registered with IANA. This situation SHOULD be avoided.
- * All registered names MUST be valid according to the XML Schema NMTOKEN data type (see [W3C.REC-xmlschema-2-20041028] Section 3.3.4). This ensures that registered names are compatible with the SWID format [SWID] where they are used.
- * Registration of vanity names SHOULD be discouraged. The requesting specification MUST provide a description of how a requested name will allow for use by multiple stakeholders.

6.2.4. Software Tag Version Scheme Values Registry

This document establishes a new registry titled "Software Tag Version Scheme Values". This registry provides index values for use as version-scheme item values in this document and version scheme names for use in [SWID].

[TO BE REMOVED: This registration should take place at the following location: <https://www.iana.org/assignments/swid>]

This registry uses the registration procedures defined in Section 6.2.1 with the following associated ranges:

Range	Registration Procedures
0-16383	Standards Action with Expert Review
16384-65535	Specification Required

Table 10: CoSWID Version Scheme Registration Procedures

Assignments MUST consist of an integer Index value, the Version Scheme Name, and a reference to the defining specification.

Initial registrations for the "Software Tag Version Scheme Values" registry are provided below, which are derived from the textual version scheme names defined in [SWID].

Index	Version Scheme Name	Specification
0	Reserved	
1	multipartnumeric	See Section 4.1
2	multipartnumeric+suffix	See Section 4.1
3	alphanumeric	See Section 4.1
4	decimal	See Section 4.1
5-16383	Unassigned	
16384	semver	See Section 4.1
16385-65535	Unassigned	

Table 11: CoSWID Version Scheme Initial Registrations

Registrations MUST conform to the expert review criteria defined in Section 6.2.3.

Designated experts MUST also ensure that newly requested entries define a value space for the corresponding version item that is unique from other previously registered entries. Note: The initial registrations violate this requirement, but are included for backwards compatibility with [SWID]. See also Section 4.1.

6.2.5. Software Tag Entity Role Values Registry

This document establishes a new registry titled "Software Tag Entity Role Values". This registry provides index values for use as entity-entry role item values in this document and entity role names for use in [SWID].

[TO BE REMOVED: This registration should take place at the following location: <https://www.iana.org/assignments/swid>]

This registry uses the registration procedures defined in Section 6.2.1 with the following associated ranges:

Range	Registration Procedures
0-127	Standards Action with Expert Review
128-255	Specification Required

Table 12: CoSWID Entity Role Registration Procedures

Assignments consist of an integer Index value, a Role Name, and a reference to the defining specification.

Initial registrations for the "Software Tag Entity Role Values" registry are provided below, which are derived from the textual entity role names defined in [SWID].

Index	Role Name	Specification
0	Reserved	
1	tagCreator	See Section 4.2
2	softwareCreator	See Section 4.2
3	aggregator	See Section 4.2
4	distributor	See Section 4.2
5	licensor	See Section 4.2
6	maintainer	See Section 4.2
7-255	Unassigned	

Table 13: CoSWID Entity Role Initial Registrations

Registrations MUST conform to the expert review criteria defined in Section 6.2.3.

6.2.6. Software Tag Link Ownership Values Registry

This document establishes a new registry titled "Software Tag Link Ownership Values". This registry provides index values for use as link-entry ownership item values in this document and link ownership names for use in [SWID].

[TO BE REMOVED: This registration should take place at the following location: <https://www.iana.org/assignments/swid>]

This registry uses the registration procedures defined in Section 6.2.1 with the following associated ranges:

Range	Registration Procedures
0-127	Standards Action with Expert Review
128-255	Specification Required

Table 14: CoSWID Link Ownership Registration Procedures

Assignments consist of an integer Index value, an Ownership Type Name, and a reference to the defining specification.

Initial registrations for the "Software Tag Link Ownership Values" registry are provided below, which are derived from the textual entity role names defined in [SWID].

Index	Ownership Type Name	Definition
0	Reserved	
1	abandon	See Section 4.3
2	private	See Section 4.3
3	shared	See Section 4.3
4-255	Unassigned	

Table 15: CoSWID Link Ownership Initial Registrations

Registrations MUST conform to the expert review criteria defined in Section 6.2.3.

6.2.7. Software Tag Link Relationship Values Registry

This document establishes a new registry titled "Software Tag Link Relationship Values". This registry provides index values for use as link-entry rel item values in this document and link ownership names for use in [SWID].

[TO BE REMOVED: This registration should take place at the following location: <https://www.iana.org/assignments/swid>]

This registry uses the registration procedures defined in Section 6.2.1 with the following associated ranges:

Range	Registration Procedures
0-32767	Standards Action with Expert Review
32768-65535	Specification Required

Table 16: CoSWID Link Relationship Registration Procedures

Assignments consist of an integer Index value, the Relationship Type Name, and a reference to the defining specification.

Initial registrations for the "Software Tag Link Relationship Values" registry are provided below, which are derived from the link relationship values defined in [SWID].

Index	Relationship Type Name	Specification
0	Reserved	
1	ancestor	See Section 4.4
2	component	See Section 4.4
3	feature	See Section 4.4
4	installationmedia	See Section 4.4
5	packageinstaller	See Section 4.4
6	parent	See Section 4.4
7	patches	See Section 4.4
8	requires	See Section 4.4
9	see-also	See Section 4.4
10	supersedes	See Section 4.4
11	supplemental	See Section 4.4
12-65535	Unassigned	

Table 17: CoSWID Link Relationship Initial Registrations

Registrations MUST conform to the expert review criteria defined in Section 6.2.3.

Designated experts MUST also ensure that a newly requested entry documents the URI schemes allowed to be used in an href associated with the link relationship and the expected resolution behavior of these URI schemes. This will help to ensure that applications processing software tags are able to interoperate when resolving resources referenced by a link of a given type.

6.2.8. Software Tag Link Use Values Registry

This document establishes a new registry titled "Software Tag Link Use Values". This registry provides index values for use as link-entry use item values in this document and link use names for use in [SWID].

[TO BE REMOVED: This registration should take place at the following location: <https://www.iana.org/assignments/swid>]

This registry uses the registration procedures defined in Section 6.2.1 with the following associated ranges:

Range	Registration Procedures
0-127	Standards Action with Expert Review
128-255	Specification Required

Table 18: CoSWID Link Use Registration Procedures

Assignments consist of an integer Index value, the Link Use Type Name, and a reference to the defining specification.

Initial registrations for the "Software Tag Link Use Values" registry are provided below, which are derived from the link relationship values defined in [SWID].

Index	Link Use Type Name	Specification
0	Reserved	
1	optional	See Section 4.5
2	required	See Section 4.5
3	recommended	See Section 4.5
4-255	Unassigned	

Table 19: CoSWID Link Use Initial Registrations

Registrations MUST conform to the expert review criteria defined in Section 6.2.3.

6.3. swid+cbor Media Type Registration

IANA is requested to add the following to the IANA "Media Types" registry [IANA.media-types].

Type name: application

Subtype name: swid+cbor

Required parameters: none

Optional parameters: none

Encoding considerations: Binary (encoded as CBOR [RFC8949]). See RFC-AAAA for details.

Security considerations: See Section 9 of RFC-AAAA.

Interoperability considerations: Applications MAY ignore any key value pairs that they do not understand. This allows backwards compatible extensions to this specification.

Published specification: RFC-AAAA

Applications that use this media type: The type is used by software asset management systems, vulnerability assessment systems, and in applications that use remote integrity verification.

Fragment Identifier Considerations: The syntax and semantics of fragment identifiers specified for "application/swid+cbor" are as specified for "application/cbor". (At publication of RFC-AAAA, there is no fragment identification syntax defined for "application/cbor".)

Additional information:

Magic number(s): if tagged, first five bytes in hex: da 53 57 49 44 (see Section 8 in RFC-AAAA)

File extension(s): coswid

Macintosh file type code(s): none

Macintosh Universal Type Identifier code: org.ietf.coswid conforms to public.data

Person & email address to contact for further information: IESG <iesg@ietf.org>

Intended usage: COMMON

Restrictions on usage: None

Author: Henk Birkholz <henk.birkholz@sit.fraunhofer.de>

Change controller: IESG

6.4. CoAP Content-Format Registration

IANA is requested to assign a CoAP Content-Format ID for the CoSWID media type in the "CoAP Content-Formats" sub-registry, from the "IETF Review or IESG Approval" space (256..999), within the "CoRE Parameters" registry [RFC7252] [IANA.core-parameters]:

Media type	Encoding	ID	Reference
application/swid+cbor	-	TBD1	RFC-AAAA

Table 20: CoAP Content-Format IDs

6.5. CBOR Tag Registration

IANA is requested to allocate a tag in the "CBOR Tags" registry [IANA.cbor-tags], preferably with the specific value requested:

Tag	Data Item	Semantics
1398229316	map	Concise Software Identifier (CoSWID) [RFC-AAAA]

Table 21: CoSWID CBOR Tag

6.6. URI Scheme Registrations

The ISO 19770-2:2015 SWID specification describes use of the "swid" and "swidpath" URI schemes, which are currently in use in implementations. This document continues this use for CoSWID. The following subsections provide registrations for these schemes in to ensure that a permanent registration exists for these schemes that is suitable for use in the SWID and CoSWID specifications.

URI schemes are registered within the "Uniform Resource Identifier (URI) Schemes" registry maintained at [IANA.uri-schemes].

6.6.1. URI-scheme swid

IANA is requested to register the URI scheme "swid". This registration request complies with [RFC7595].

Scheme name:
swid

Status:
Permanent

Applications/protocols that use this scheme name:
Applications that require Software-IDs (SWIDs) or Concise Software-IDs (CoSWIDs); see Section 5.1 of RFC-AAAA.

Contact:
IETF Chair <chair@ietf.org>

Change controller:
IESG <iesg@ietf.org>

Reference:
Section 5.1 in RFC-AAAA

6.6.2. URI-scheme swidpath

IANA is requested to register the URI scheme "swidpath". This registration request complies with [RFC7595].

Scheme name:
swidpath

Status:
Permanent

Applications/protocols that use this scheme name:
Applications that require Software-IDs (SWIDs) or Concise Software-IDs (CoSWIDs); see Section 5.2 of RFC-AAAA.

Contact:
IETF Chair <chair@ietf.org>

Change controller:
IESG <iesg@ietf.org>

Reference:
Section 5.2 in RFC-AAAA

6.7. CoSWID Model for use in SWIMA Registration

The Software Inventory Message and Attributes (SWIMA) for PA-TNC specification [RFC8412] defines a standardized method for collecting an endpoint device's software inventory. A CoSWID can provide evidence of software installation which can then be used and exchanged with SWIMA. This registration adds a new entry to the IANA "Software Data Model Types" registry defined by [RFC8412] [IANA.pa-tnc-parameters] to support CoSWID use in SWIMA as follows:

Pen: 0

Integer: TBD2

Name: Concise Software Identifier (CoSWID)

Reference: RFC-AAAA

Deriving Software Identifiers:

A Software Identifier generated from a CoSWID tag is expressed as a concatenation of the form in [RFC5234] as follows:

TAG_CREATOR_REGID "_" "_" UNIQUE_ID

Where TAG_CREATOR_REGID is the reg-id item value of the tag's entity item having the role value of 1 (corresponding to "tag creator"), and the UNIQUE_ID is the same tag's tag-id item. If the tag-id item's value is expressed as a 16-byte binary string, the UNIQUE_ID MUST be represented using the UUID string representation defined in [RFC4122] including the "urn:uuid:" prefix.

The TAG_CREATOR_REGID and the UNIQUE_ID are connected with a double underscore (_), without any other connecting character or whitespace.

7. Signed CoSWID Tags

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.

Signing CoSWID tags follows the procedures defined in CBOR Object Signing and Encryption [I-D.ietf-cose-rfc8152bis-struct]. A CoSWID tag MUST be wrapped in a COSE Signature structure, either COSE_Sign1 or COSE_Sign. In the first case, a Single Signer Data Object (COSE_Sign1) contains a single signature and MUST be signed by the tag creator. The following CDDL specification defines a restrictive subset of COSE header parameters that MUST be used in the protected header in this case.

```
<CODE BEGINS>
COSE-Sign1-coswid<payload> = [
    protected: bstr .cbor protected-signed-coswid-header,
    unprotected: unprotected-signed-coswid-header,
    payload: bstr .cbor payload,
    signature: bstr,
]

cose-label = int / tstr
cose-values = any

protected-signed-coswid-header = {
    1 => int,                ; algorithm identifier
    3 => "application/swid+cbor",
    * cose-label => cose-values,
}

unprotected-signed-coswid-header = {
    * cose-label => cose-values,
}
<CODE ENDS>
```

The COSE_Sign structure allows for more than one signature, one of which MUST be issued by the tag creator, to be applied to a CoSWID tag and MAY be used. The corresponding usage scenarios are domain-specific and require well-specified application guidance.

```
<CODE BEGINS>
COSE-Sign-coswid<payload> = [
    protected: bstr .cbor protected-signed-coswid-header1,
    unprotected: unprotected-signed-coswid-header,
    payload: bstr .cbor payload,
    signature: [ * COSE_Signature ],
]

protected-signed-coswid-header1 = {
    3 => "application/swid+cbor",
    * cose-label => cose-values,
}

protected-signature-coswid-header = {
    1 => int,                      ; algorithm identifier
    * cose-label => cose-values,
}

unprotected-sign-coswid-header = {
    * cose-label => cose-values,
}

COSE_Signature = [
    protected: bstr .cbor protected-signature-coswid-header,
    unprotected: unprotected-sign-coswid-header,
    signature : bstr
]
<CODE ENDS>
```

Additionally, the COSE Header counter signature MAY be used as an attribute in the unprotected header map of the COSE envelope of a CoSWID [I-D.ietf-cose-countersign]. The application of counter signing enables second parties to provide a signature on a signature allowing for a proof that a signature existed at a given time (i.e., a timestamp).

A CoSWID MUST be signed, using the above mechanism, to protect the integrity of the CoSWID tag. See the security considerations (in Section 9) for more information on why a signed CoSWID is valuable in most cases.

8. CBOR-Tagged CoSWID Tags

This specification allows for tagged and untagged CBOR data items that are CoSWID tags. Consecutively, the CBOR tag for CoSWID tags defined in Table 21 SHOULD be used in conjunction with CBOR data items that are a CoSWID tags. Other CBOR tags MUST NOT be used with a CBOR data item that is a CoSWID tag. If tagged, both signed and unsigned CoSWID tags MUST use the CoSWID CBOR tag. In case a signed CoSWID is tagged, a CoSWID CBOR tag MUST be appended before the COSE envelope whether it is a COSE_Untagged_Message or a COSE_Tagged_Message. In case an unsigned CoSWID is tagged, a CoSWID CBOR tag MUST be appended before the CBOR data item that is the CoSWID tag.

```
<CODE BEGINS>
coswid = unsigned-coswid / signed-coswid
unsigned-coswid = concise-swid-tag / tagged-coswid<concise-swid-tag>
signed-coswid1 = signed-coswid-for<unsigned-coswid>
signed-coswid = signed-coswid1 / tagged-coswid<signed-coswid1>

tagged-coswid<T> = #6.1398229316(T)

signed-coswid-for<payload> = #6.18(COSE-Sign1-coswid<payload>)
    / #6.98(COSE-Sign-coswid<payload>)
<CODE ENDS>
```

This specification allows for a tagged CoSWID tag to reside in a COSE envelope that is also tagged with a CoSWID CBOR tag. In cases where a tag creator is not a signer (e.g., hand-offs between entities in a trusted portion of a supply-chain), retaining CBOR tags attached to unsigned CoSWID tags can be of great use. Nevertheless, redundant use of tags SHOULD be avoided when possible.

9. Security Considerations

The following security considerations for use of CoSWID tags focus on:

- * ensuring the integrity and authenticity of a CoSWID tag
- * the application of CoSWID tags to address security challenges related to unmanaged or unpatched software
- * reducing the potential for unintended disclosure of a device's software load

A tag is considered "authoritative" if the CoSWID tag was created by the software provider. An authoritative CoSWID tag contains information about a software component provided by the supplier of the software component, who is expected to be an expert in their own software. Thus, authoritative CoSWID tags can represent authoritative information about the software component. The degree to which this information can be trusted depends on the tag's chain of custody and the ability to verify a signature provided by the supplier if present in the CoSWID tag. The provisioning and validation of CoSWID tags are handled by local policy and is outside the scope of this document.

A signed CoSWID tag (see Section 7) whose signature has been validated can be relied upon to be unchanged since it was signed. By contrast, the data contained in unsigned tags can be altered by any user or process with write-access to the tag. To support signature validation, there is the need to associate the right key with the software provider or party originating the signature in a secure way. This operation is application specific and needs to be addressed by the application or a user of the application; a specific approach for which is out-of-scope for this document.

When an authoritative tag is signed, the originator of the signature can be verified. A trustworthy association between the signature and the originator of the signature can be established via trust anchors. A certification path between a trust anchor and a certificate including a public key enabling the validation of a tag signature can realize the assessment of trustworthiness of an authoritative tag. Verifying that the software provider is the signer is a different matter. This requires an association between the signature and the tag's entity item associated corresponding to the software provider. No mechanism is defined in this draft to make this association; therefore, this association will need to be handled by local policy. As always, the validity of a signature does not imply veracity of the signed statements: anyone can sign assertions such that the software is from a specific software-creator or that a specific persistent-id applies; policy needs to be applied to evaluate these statements and to determine their suitability for a specific use.

Loss of control of signing credentials used to sign CoSWID tags would create doubt about the authenticity and integrity of any CoSWID tags signed using the compromised keys. In such cases, the legitimate tag signer (namely, the software provider for an authoritative CoSWID tag) can employ uncompromised signing credentials to create a new signature on the original tag. The tag version number would not be incremented since the tag itself was not modified. Consumers of CoSWID tags would need to validate the tag using the new credentials and would also need to make use of revocation information available

for the compromised credentials to avoid validating tags signed with them. The process for doing this is beyond the scope of this specification.

The CoSWID format allows the use of hash values without an accompanying hash algorithm identifier. This exposes the tags to some risk of cross-algorithm attacks. We believe that this can become a practical problem only if some implementations allow the use of insecure hash algorithms. Since it may not become known immediately when an algorithm becomes insecure, this leads to a strong recommendation to only include support for hash algorithms that are generally considered secure, and not just marginally so.

CoSWID tags are intended to contain public information about software components and, as such, the contents of a CoSWID tag (as opposed to the set of tags that apply to the endpoint, see below) does not need to be protected against unintended disclosure on an endpoint. Converse, generators of CoSWID tags need to ensure that only public information is disclosed. Entitlement Keys are an example for information where particular care is required; tag authors are advised not to record unprotected, private software license keys in this field.

CoSWID tags are intended to be easily discoverable by authorized applications and users on an endpoint in order to make it easy to determine the tagged software load. Access to the collection of an endpoint's CoSWID tags needs to be appropriately controlled to authorized applications and users using an appropriate access control mechanism.

Since the tag-id of a CoSWID tag can be used as a global index value, failure to ensure the tag-id's uniqueness can cause collisions or ambiguity in CoSWID tags that are retrieved or processed using this identifier. CoSWID is designed to not require a registry of identifiers. As a result, CoSWID requires the tag creator to employ a method of generating a unique tag identifier. Specific methods of generating a unique identifier are beyond the scope of this specification. A collision in tag-ids may result in false positives/negatives in software integrity checks or mis-identification of installed software, undermining CoSWID use cases such as vulnerability identification, software inventory, etc. If such a collision is detected, then the tag consumer may want to contact the maintainer of the CoSWID to have them issue a correction addressing the collision; however, this also discloses to the maintainer that the consumer has the other tag with the given tag-id in their database. More generally speaking, a tag consumer needs to be robust against such collisions lest the collision become a viable attack vector.

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

As CoSWID tags do not expire, inhibiting new CoSWID tags from reaching an intended consumer would render that consumer stuck with outdated information, potentially leaving associated vulnerabilities or weaknesses unmitigated. Therefore, a CoSWID tag consumer should actively check for updated tag-versions via more than one means.

This specification makes use of relative paths (e.g., filesystem paths) in several places. A signed CoSWID tag cannot make use of these to derive information that is considered to be covered under the signature. Typically, relative file system paths will be used to identify targets for an installation, not sources of tag information.

Any report of an endpoint's CoSWID 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, CoSWID tags are designed to be easily discoverable by authorized applications and users on 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. For this reason, it is important to protect the confidentiality of CoSWID tag information that has been collected from an endpoint in transit and at rest, not because those tags individually contain sensitive information, but because the collection of CoSWID tags and their association with an endpoint reveals information about that endpoint's attack surface.

Finally, both the ISO-19770-2:2015 XML schema SWID definition and the CoSWID CDDL specification allow for the construction of "infinite" tags with link item loops or tags that contain malicious content with the intent of creating non-deterministic states during validation or processing of those tags. While software providers are unlikely to do this, CoSWID tags can be created by any party and the CoSWID tags collected from an endpoint could contain a mixture of vendor and non-vendor created tags. For this reason, a CoSWID tag might contain potentially malicious content. Input sanitization, loop detection, and signature verification are ways that implementations can address this concern.

More generally speaking, the security considerations of [RFC8949], [I-D.ietf-cose-rfc8152bis-struct], and [I-D.ietf-cose-countersign] apply.

10. Privacy Consideration

As noted in Section 9, collected information about an endpoint's software load, such as what might be represented by an endpoint's CoSWID tag collection, could be used to identify vulnerable software for attack. Collections of endpoint software information also can have privacy implications for users. The set of application a user installs can give clues to personal matters such as political affiliation, banking and investments, gender, sexual orientation, medical concerns, etc. While the collection of CoSWID tags on an endpoint wouldn't increase the privacy risk (since a party able to view those tags could also view the applications themselves), if

those CoSWID tags are gathered and stored in a repository somewhere, visibility into the repository now also gives visibility into a user's application collection. For this reason, repositories of collected CoSWID tags not only need to be protected against collection by malicious parties, but even authorized parties will need to be vetted and made aware of privacy responsibilities associated with having access to this information. Likewise, users should be made aware that their software inventories are being collected from endpoints. Furthermore, when collected and stored by authorized parties or systems, the inventory data needs to be protected as both security and privacy sensitive information.

11. Change Log

This section is to be removed before publishing as an RFC.

[THIS SECTION TO BE REMOVED BY THE RFC EDITOR.]

Changes from version 12 to version 14:

- * Moved key identifier to protected COSE header
- * Fixed index reference for hash
- * Removed indirection of CDDL type definition for filesystem-item
- * Fixed quantity of resource and process
- * Updated resource-collection
- * Renamed socket name in software-meta to be consistent in naming
- * Aligned excerpt examples in I-D text with full CDDL
- * Fixed titles where title was referring to group instead of map
- * Added missing date in SEMVER
- * Fixed root cardinality for file and directory, etc.
- * Transformed path-elements-entry from map to group for re-usability
- * Scrubbed IANA Section
- * Removed redundant supplemental rule
- * Aligned discrepancy with ISO spec.

- * Addressed comments on typos.
- * Fixed kramdown nits and BCP reference.
- * Addressed comments from WGLC reviewers.

Changes in version 12:

- * Addressed a bunch of minor editorial issues based on WGLC feedback.
- * Added text about the use of UTF-8 in CoSWID.
- * Adjusted tag-id to allow for a UUID to be provided as a bstr.
- * Cleaned up descriptions of index ranges throughout the document, removing discussion of 8 bit, 16 bit, etc.
- * Adjusted discussion of private use ranges to use negative integer values and to be more clear throughout the document.
- * Added discussion around resolving overlapping value spaces for version schemes.
- * Added a set of expert review criteria for new IANA registries created by this document.
- * Added new registrations for the "swid" and "swidpath" URI schemes, and for using CoSWID with SWIMA.

Changes from version 03 to version 11:

- * Reduced representation complexity of the media-entry type and removed the Section describing the older data structure.
- * Added more signature schemes from COSE
- * Included a minimal required set of normative language
- * Reordering of attribute name to integer label by priority according to semantics.
- * Added an IANA registry for CoSWID items supporting future extension.
- * Cleaned up IANA registrations, fixing some inconsistencies in the table labels.

- * Added additional CDDL sockets for resource collection entries providing for additional extension points to address future SWID/CoSWID extensions.
- * Updated Section on extension points to address new CDDL sockets and to reference the new IANA registry for items.
- * Removed unused references and added new references to address placeholder comments.
- * Added table with semantics for the link ownership item.
- * Clarified language, made term use more consistent, fixed references, and replacing lowercase RFC2119 keywords.

Changes from version 02 to version 03:

- * Updated core CDDL including the CDDL design pattern according to RFC 8428.

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

Changes from version 00 to version 01:

- * Ambiguity between evidence and payload eliminated by introducing explicit members (while still
- * allowing for "empty" SWID tags)
- * Added a relatively restrictive COSE envelope using cose_sign1 to define signed CoSWID (single signer only, at the moment)

- * Added a definition how to encode hashes that can be stored in the any-member using existing IANA tables to reference hash-algorithms

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

Changes from version 06 to version 07:

- * Added type choices/enumerations based on textual definitions in 19770-2:2015
- * Added value registry request
- * Added media type registration request
- * Added content format registration request
- * Added CBOR tag registration request
- * Removed RIM appendix to be addressed in complementary draft
- * Removed CWT appendix
- * Flagged firmware resource collection appendix for revision
- * Made use of terminology more consistent
- * Better defined use of extension points in the CDDL
- * Added definitions for indexed values
- * Added IANA registry for Link use indexed values

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

12. References

12.1. Normative References

- [BCP178] Saint-Andre, P., Crocker, D., and M. Nottingham, "Deprecating the "X-" Prefix and Similar Constructs in Application Protocols", BCP 178, RFC 6648, DOI 10.17487/RFC6648, June 2012, <<https://www.rfc-editor.org/info/rfc6648>>.
- [BCP26] Cotton, M., Leiba, B., and T. Narten, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 8126, DOI 10.17487/RFC8126, June 2017, <<https://www.rfc-editor.org/info/rfc8126>>.
- [I-D.ietf-cose-countersign] Schaad, J. and R. Housley, "CBOR Object Signing and Encryption (COSE): Countersignatures", Work in Progress, Internet-Draft, draft-ietf-cose-countersign-05, 23 June 2021, <<https://www.ietf.org/archive/id/draft-ietf-cose-countersign-05.txt>>.
- [I-D.ietf-cose-rfc8152bis-struct] Schaad, J., "CBOR Object Signing and Encryption (COSE): Structures and Process", Work in Progress, Internet-Draft, draft-ietf-cose-rfc8152bis-struct-15, 1 February 2021, <<https://www.ietf.org/archive/id/draft-ietf-cose-rfc8152bis-struct-15.txt>>.

- [IANA.cbor-tags]
IANA, "Concise Binary Object Representation (CBOR) Tags",
<<http://www.iana.org/assignments/cbor-tags>>.
- [IANA.core-parameters]
IANA, "Constrained RESTful Environments (CoRE)
Parameters",
<<http://www.iana.org/assignments/core-parameters>>.
- [IANA.media-types]
IANA, "Media Types",
<<http://www.iana.org/assignments/media-types>>.
- [IANA.named-information]
IANA, "Named Information",
<<http://www.iana.org/assignments/named-information>>.
- [IANA.pa-tnc-parameters]
IANA, "Posture Attribute (PA) Protocol Compatible with
Trusted Network Connect (TNC) Parameters",
<<http://www.iana.org/assignments/pa-tnc-parameters>>.
- [IANA.uri-schemes]
IANA, "Uniform Resource Identifier (URI) Schemes",
<<http://www.iana.org/assignments/uri-schemes>>.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate
Requirement Levels", BCP 14, RFC 2119,
DOI 10.17487/RFC2119, March 1997,
<<https://www.rfc-editor.org/info/rfc2119>>.
- [RFC3629] Yergeau, F., "UTF-8, a transformation format of ISO
10646", STD 63, RFC 3629, DOI 10.17487/RFC3629, November
2003, <<https://www.rfc-editor.org/info/rfc3629>>.
- [RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform
Resource Identifier (URI): Generic Syntax", STD 66,
RFC 3986, DOI 10.17487/RFC3986, January 2005,
<<https://www.rfc-editor.org/info/rfc3986>>.
- [RFC5198] Klensin, J. and M. Padlipsky, "Unicode Format for Network
Interchange", RFC 5198, DOI 10.17487/RFC5198, March 2008,
<<https://www.rfc-editor.org/info/rfc5198>>.
- [RFC5234] Crocker, D., Ed. and P. Overell, "Augmented BNF for Syntax
Specifications: ABNF", STD 68, RFC 5234,
DOI 10.17487/RFC5234, January 2008,
<<https://www.rfc-editor.org/info/rfc5234>>.

- [RFC5646] Phillips, A., Ed. and M. Davis, Ed., "Tags for Identifying Languages", BCP 47, RFC 5646, DOI 10.17487/RFC5646, September 2009, <<https://www.rfc-editor.org/info/rfc5646>>.
- [RFC5890] Klensin, J., "Internationalized Domain Names for Applications (IDNA): Definitions and Document Framework", RFC 5890, DOI 10.17487/RFC5890, August 2010, <<https://www.rfc-editor.org/info/rfc5890>>.
- [RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, DOI 10.17487/RFC7252, June 2014, <<https://www.rfc-editor.org/info/rfc7252>>.
- [RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <<https://www.rfc-editor.org/info/rfc8174>>.
- [RFC8288] Nottingham, M., "Web Linking", RFC 8288, DOI 10.17487/RFC8288, October 2017, <<https://www.rfc-editor.org/info/rfc8288>>.
- [RFC8412] Schmidt, C., Haynes, D., Coffin, C., Waltermire, D., and J. Fitzgerald-McKay, "Software Inventory Message and Attributes (SWIMA) for PA-TNC", RFC 8412, DOI 10.17487/RFC8412, July 2018, <<https://www.rfc-editor.org/info/rfc8412>>.
- [RFC8610] Birkholz, H., Vigano, C., and C. Bormann, "Concise Data Definition Language (CDDL): A Notational Convention to Express Concise Binary Object Representation (CBOR) and JSON Data Structures", RFC 8610, DOI 10.17487/RFC8610, June 2019, <<https://www.rfc-editor.org/info/rfc8610>>.
- [RFC8949] Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", STD 94, RFC 8949, DOI 10.17487/RFC8949, December 2020, <<https://www.rfc-editor.org/info/rfc8949>>.
- [SAM] "Information technology - Software asset management - Part 5: Overview and vocabulary", ISO/IEC 19770-5:2015, 15 November 2013.
- [SEMVER] Preston-Werner, T., "Semantic Versioning 2.0.0", <<https://semver.org/spec/v2.0.0.html>>.

- [SWID] "Information technology - Software asset management - Part 2: Software identification tag", ISO/IEC 19770-2:2015, 1 October 2015.
- [UNSPSC] "United Nations Standard Products and Services Code", 26 October 2020, <<https://www.unspsc.org/>>.
- [W3C.REC-css3-mediaqueries-20120619]
Rivoal, F., "Media Queries", World Wide Web Consortium Recommendation REC-css3-mediaqueries-20120619, 19 June 2012, <<https://www.w3.org/TR/2012/REC-css3-mediaqueries-20120619>>.
- [W3C.REC-xmlschema-2-20041028]
Biron, P. and A. Malhotra, "XML Schema Part 2: Datatypes Second Edition", World Wide Web Consortium Recommendation REC-xmlschema-2-20041028, 28 October 2004, <<https://www.w3.org/TR/2004/REC-xmlschema-2-20041028>>.
- [W3C.REC-xpath20-20101214]
Berglund, A., Boag, S., Chamberlin, D., Fernandez, M., Kay, M., Robie, J., and J. Simeon, "XML Path Language (XPath) 2.0 (Second Edition)", World Wide Web Consortium Recommendation REC-xpath20-20101214, 14 December 2010, <<https://www.w3.org/TR/2010/REC-xpath20-20101214>>.

12.2. Informative References

- [CamelCase]
"UpperCamelCase", 29 August 2014, <<http://wiki.c2.com/?CamelCase>>.
- [I-D.ietf-rats-architecture]
Birkholz, H., Thaler, D., Richardson, M., Smith, N., and W. Pan, "Remote Attestation Procedures Architecture", Work in Progress, Internet-Draft, draft-ietf-rats-architecture-15, 8 February 2022, <<https://www.ietf.org/archive/id/draft-ietf-rats-architecture-15.txt>>.
- [KebabCase]
"KebabCase", 18 December 2014, <<http://wiki.c2.com/?KebabCase>>.
- [RFC3444] Pras, A. and J. Schoenwaelder, "On the Difference between Information Models and Data Models", RFC 3444, DOI 10.17487/RFC3444, January 2003, <<https://www.rfc-editor.org/info/rfc3444>>.

- [RFC4122] Leach, P., Mealling, M., and R. Salz, "A Universally Unique Identifier (UUID) URN Namespace", RFC 4122, DOI 10.17487/RFC4122, July 2005, <<https://www.rfc-editor.org/info/rfc4122>>.
- [RFC7595] Thaler, D., Ed., Hansen, T., and T. Hardie, "Guidelines and Registration Procedures for URI Schemes", BCP 35, RFC 7595, DOI 10.17487/RFC7595, June 2015, <<https://www.rfc-editor.org/info/rfc7595>>.
- [RFC8322] Field, J., Banghart, S., and D. Waltermire, "Resource-Oriented Lightweight Information Exchange (ROLIE)", RFC 8322, DOI 10.17487/RFC8322, February 2018, <<https://www.rfc-editor.org/info/rfc8322>>.
- [RFC8520] Lear, E., Droms, R., and D. Romascanu, "Manufacturer Usage Description Specification", RFC 8520, DOI 10.17487/RFC8520, March 2019, <<https://www.rfc-editor.org/info/rfc8520>>.
- [SWID-GUIDANCE]
Waltermire, D., Cheikes, B.A., Feldman, L., and G. Witte, "Guidelines for the Creation of Interoperable Software Identification (SWID) Tags", NISTIR 8060, April 2016, <<https://doi.org/10.6028/NIST.IR.8060>>.
- [X.1520] "Recommendation ITU-T X.1520 (2014), Common vulnerabilities and exposures", 20 April 2011.

Acknowledgments

This document draws heavily on the concepts defined in the ISO/IEC 19770-2:2015 specification. The authors of this document are grateful for the prior work of the 19770-2 contributors.

We are also grateful for the careful reviews provided by the IESG reviewers. Special thanks go to Benjamin Kaduk.

Contributors

Carsten Bormann
Universität Bremen TZI
Postfach 330440
D-28359 Bremen
Germany
Phone: +49-421-218-63921
Email: cabo@tzi.org

Carsten Bormann contributed to the CDDL specifications and the IANA considerations.

Authors' Addresses

Henk Birkholz
Fraunhofer SIT
Rheinstrasse 75
64295 Darmstadt
Germany
Email: henk.birkholz@sit.fraunhofer.de

Jessica Fitzgerald-McKay
National Security Agency
9800 Savage Road
Ft. Meade, Maryland
United States of America
Email: jmfitz2@cyber.nsa.gov

Charles Schmidt
The MITRE Corporation
202 Burlington Road
Bedford, Massachusetts 01730
United States of America
Email: cmschmidt@mitre.org

David Waltermire
National Institute of Standards and Technology
100 Bureau Drive
Gaithersburg, Maryland 20877
United States of America
Email: david.waltermire@nist.gov