SACM Working Group
Internet-Draft
Intended status: InformationalNational Institute of Standards and Techno
Expires: November 4, 2017

D. Waltermire
S. Banghart

May 3, 2017

Definition of the ROLIE Software Descriptor Extension draft-banghart-sacm-rolie-softwaredescriptor-01

Abstract

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

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on November 4, 2017.

Copyright Notice

Table of Contents

1. Introduction

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

an installable software package, or

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

Software descriptor information includes identifying, versioning, software creation and publication, and file artifact information. Software descriptor information provides data about what might be

installed, but doesn't describe where or how a specific software installation is installed, configured, or executed.

Some possible use cases for Software descriptor information include:

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

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

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

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

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

2. Terminology

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

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

3. New information-types

This document defines the following information type:

3.1. The "software-descriptor" information type

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

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

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

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

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

- Naming information: IDs and names that aid in the identification of a piece of software
- Version and patching information: Version numbers, patch identifiers, or other information that
- Vendor and source information: Includes where the software was developed or distributed from, as well as where the software installation media may be located.
- Payload and file information: information that describes or enumerates the files and folders that make up the piece of software, and information about those files.
- Descriptive information and data: Any information that otherwise characterizes a piece of software, such as libraries, runtime environments, target OSes, intended purpose or audience, etc.

Note again that this list is not exhaustive, any information that in is the abstract realm of an incident should be classified under this information-type.

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

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

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

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

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

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

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

6.1. Use of the atom:link element

This section defines the requirements around the use of atom:links in Entries. Each relationship should be named,described, and given a requirement level.

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

Table 1: Link Relations for Resource-Oriented Lightweight Indicator Exchange

6.2. Use of the rolie:format element

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

6.2.1. The ISO SWID 2016 format

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

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

6.2.2. The Concise SWID format

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

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

6.3. Use of the rolie:property element

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

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

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

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

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

6.4. IANA Considerations

6.4.1. incident information-type

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

The entry is as follows:

name: software-descriptor

index: TBD

reference: This document, Section 3.1

6.4.2. swd:id property

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

The entry is as follows:

name: property:swd:id

Extension IRI: urn:ietf:params:rolie:property:swd:id

Reference: This document, Section 6.3.1

Subregistry: None

6.4.3. swd:swname property

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

The entry is as follows:

name: property:swd:swname

Extension IRI: urn:ietf:params:rolie:property:swd:swname

Reference: This document, Section 6.3.2

Subregistry: None

6.5. Security Considerations

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

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

These "authoritative" sources of SWD content should consider additional security for their ROLIE repository beyond the typical recommendations, as the central importance of the repository is likely to make it a target.

Version information is often represented differently across manufacturers and even across product releases. If using SWD version information for low fault tolerance comparisons and searches, care should be taken that the correct version scheme is being utilized.

7. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <http://www.rfc-editor.org/info/rfc2119>.

[RFC5070] Danyliw, R., Meijer, J., and Y. Demchenko, "The Incident Object Description Exchange Format", RFC 5070, DOI 10.17487/RFC5070, December 2007, <http://www.rfc-editor.org/info/rfc5070>.

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

Appendix A. Schema

This document does not require any schema extensions.

Appendix B. Examples of Use

Use of this extension in a ROLIE repository will not typically change that repo’s operation. As such, the general examples provided by the ROLIE core document would serve as examples. Provided below is a sample SWD ROLIE entry:

```
<?xml version="1.0" encoding="UTF-8"?>
<entry xmlns="http://www.w3.org/2005/Atom"
  xmlns:rolie="urn:ietf:params:xml:ns:rolie-1.0">
  <id>dd786dba-88e6-440b-9158-b8fae67ef67c</id>
  <title>Sample Software Descriptor</title>
  <published>2015-08-04T18:13:51.0Z</published>
  <updated>2015-08-05T18:13:51.0Z</updated>
  <summary>A descriptor for a piece of software published by this
  organization. </summary>
  <link rel="self" href="http://www.example.org/provider/SWD/123456"/>
  <category
      scheme="urn:ietf:params:rolie:category:information-type"
      term="software-descriptor"/>
  <rolie:format ns="urn:example:COSWID"/>
  <content type="application/xml"
    src="http://www.example.org/provider/SWD/123456/data"/>
</entry>
```

Authors' Addresses

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

Email: david.waltermire@nist.gov

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

Email: stephen.banghart@nist.gov

SACM
Internet-Draft
Intended status: Standards Track
Expires: May 4, 2017

C. Coffin
D. Haynes
C. Schmidt
The MITRE Corporation
J. Fitzgerald-McKay
Department of Defense

October 31, 2016

Software Inventory Message and Attributes (SWIMA) for PA-TNC
draft-coffin-sacm-nea-swid-patnc-03

Abstract

This document specifies the Software Inventory Message and Attributes for PA-TNC. It extends the PA-TNC specification [RFC5792] by providing specific attributes and message exchanges to allow endpoints to report their installed software inventory information to a NEA server (as described in [RFC5209]).

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on May 4, 2017.

Table of Contents

1. Introduction

Possession of a list of an endpoint's installed software is very useful in understanding and maintaining the security state of an enterprise. For example, if an enterprise policy requires the presence of certain pieces of software and/or prohibits the presence of other software, reported software installation inventory lists can be used to indicate compliance or non-compliance with these requirements. Software installation inventories and the patch level of the identified software can be compared to vulnerability or threat alerts to determine an endpoint's exposure to attack. All of these uses make an understanding of an endpoint's installed software inventory highly useful to NEA Servers and other enterprise security applications.

There is a need for a standardized method for exchanging software inventory that carries a software identifier (a unique identifier associated with a specific software product and version thereof). In some cases, it may also be necessary to convey information that characterizes this product (i.e., provides metadata about the product beyond its identifier) as well as observable installation information. These "Messages and Attributes" would enable software identification, installation, and characterization information to be provided for any software installed on any endpoint that supports this specification.

To that end, this specification defines a new set of PA-TNC attributes, carried over PA-TNC messages, which are used to communicate requests for software information and software events, and for conveying that information back to a NEA Server.

This specification is designed only to report software that is installed on an endpoint. In particular, it does not monitor or report information about what software is running on the endpoint. Likewise, it is not intended to report individual files, libraries, installation packages, or similar artifacts. While all of this

information has its uses, this information requires different metadata and different methods of monitoring the endpoint. As a result, this specification focuses solely on installed software, leaving reporting of other classes of endpoint information to other specifications.

1.1. Network Endpoint Assessment (NEA)

The Network Endpoint Assessment (NEA) Working Group defines an open solution architecture that enables network operators to collect and utilize information about endpoint configuration and state. This information can be used to enforce policies, monitor endpoint health, and for many other activities. Information about the software present on an endpoint is an important consideration for such activities. Such information can come from multiple sources, including tag files (such as ISO SWID tags [SWID], reports from third party inventory tools, output from package managers, and other sources. The attributes defined in this document are used to communicate software inventory evidence, collected from a range of possible sources, from the posture collector on the endpoint to the posture validator on a NEA Server using the PA-TNC interface, as shown in Figure 1 below.

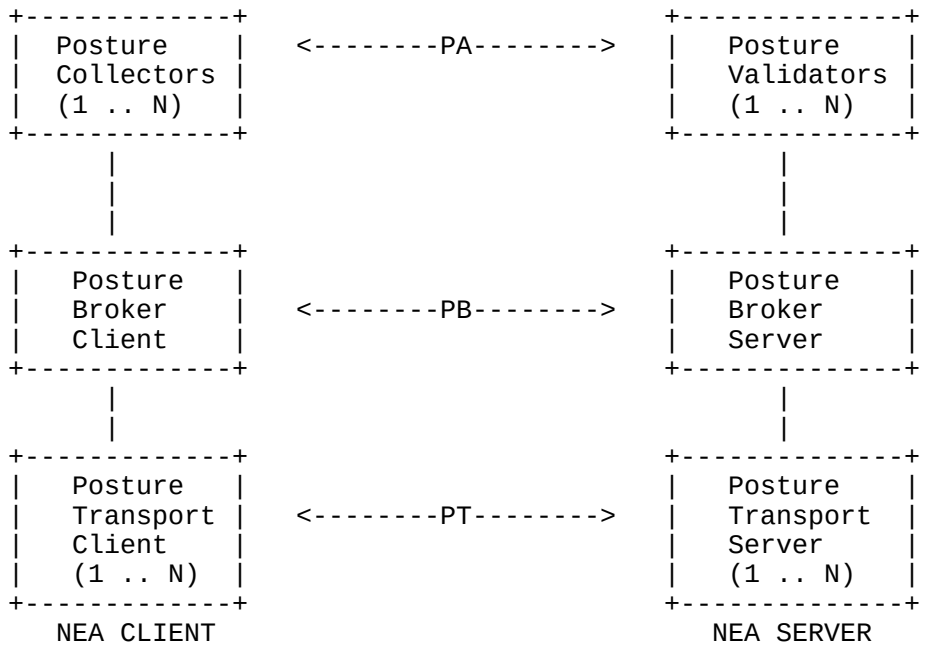


Figure 1: NEA Reference Model

Before reading this specification any further, the reader should review and understand the NEA reference architecture as described in the Network Endpoint Assessment (NEA): Overview and Requirements [RFC5209]. The reader should also understand the capabilities and requirements common to PA-TNC interfaces as defined in RFC 5792 [RFC5792].

This document is based on standards published by the Trusted Computing Group’s Trusted Network Communications (TNC) workgroup. The TNC and NEA architectures are interoperable and the following components are equivalent:

TNC Component	NEA Component
Integrity Measurement Verifier (IMV)	Posture Validator
Integrity Measurement Collector (IMC)	Posture Collector
TNC Server (TNCS)	Posture Broker Server
TNC Client (TNCC)	Posture Broker Client

Table 1: Equivalent components in TNC and NEA architectures

1.2. Keywords

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this specification are to be interpreted as described in RFC 2119 [RFC2119]. This specification does not distinguish blocks of informative comments and normative requirements. Therefore, for the sake of clarity, note that lower case instances of must, should, etc. do not indicate normative requirements.

1.3. Definitions

This section defines terms with special meaning within this document.

SW-PC - A Posture Collector (PC) that conforms to this specification. Note that such a posture collector might also support other PA-TNC exchanges beyond Software Inventory Message and Attributes for PA-TNC.

SW-PV - A Posture Validator (PV) that conforms to this specification. Note that such a posture verifier might also support other PA-TNC exchanges beyond Software Inventory Message and Attributes for PA-TNC.

SW Attribute - This is a PA-TNC attribute (as defined in RFC 5792 [RFC5792] extension for conveying software inventory information. This specification defines several new attribute types.

Endpoint’s Software Inventory Evidence Collection - The set of information regarding the set of software installed on an endpoint. An endpoint’s software inventory evidence collection might include information created by or derived from multiple sources, including but not limited to SWID tag files deposited on the file system during software installation, information generated to report output from

software discovery tools, and information dynamically generated by a software or package management system on an endpoint.

Software Inventory Evidence Record - The endpoint's Software Inventory Evidence Collection is composed of "records". Each record corresponds to one installed instance of a particular software product as reported by some data source. It is possible for a single installed instance to have multiple software inventory evidence records in an endpoint's Software Inventory Evidence Collection - this can happen if multiple sources all report the same software installation instance.

Software Identifier - A string associated with a specific version of a specific software product. Each supported Software Inventory Message and Attributes for PA-TNC data model has its own rules for how a Software Identifier (see Section 3.2.1) is derived from the representation of the given software product using that data model, and different sources for this information might populate relevant information differently. As such, while each Software Identifier uniquely identifies a specific software product, the same software product might be associated with multiple Software Identifiers reflecting differences between different information sources and supported data models.

2. Background

2.1. Supported Use Cases

This section describes the Software Inventory Message and Attributes for PA-TNC use cases supported by this specification. The primary use of exchanging software inventory information over the PA-TNC interface is to enable a challenger (e.g. NEA Server) to obtain inventory evidence about some system in a way that conforms to NEA procedures and expressed using a standard format. Collected software information can support a range of security activities including determining whether an endpoint is permitted to connect to the enterprise, determining which endpoints contain software that requires patching, and similar activities.

2.1.1. Use Software Inventory as a Factor in Determining Endpoint Access

Some enterprises might define security policies that require connected endpoints to have certain pieces of security software installed. By contrast, some security policies might prevent access by endpoints that have certain prohibited pieces of software installed, such as applications that pose known security risks. To support such policies, the NEA Server needs to collect evidence

indicating the software inventory of an endpoint that is seeking to initiate or continue connectivity to the enterprise.

Software Inventory Message and Attributes for PA-TNC facilitates policy decisions that consider an endpoint's software inventory by providing the NEA Server with software inventory information from the endpoint. The SW-PC can provide a complete or partial inventory to the SW-PV as required to determine policy compliance. The SW-PV can then use this as evidence of compliance or non-compliance with enterprise policy.

2.1.2. Maintain a Central Repository Reflecting an Endpoint's Software Inventory

Many tools can use information about an endpoint's software inventory to monitor and enforce the security of an enterprise. For example, a software patching service can use an endpoint's software inventory to determine whether certain endpoints have software that requires patching. A vulnerability management tool might identify endpoints with known vulnerabilities (patched or otherwise) and use this to gauge enterprise exposure to attack. A license management tool might verify that all copies of a particular piece of software are accounted for within the enterprise. The presence of a central repository representing a real-time understanding of each endpoint's software inventory facilitates all such activities. Using a central repository that can ensure the freshness of its collected information is generally more efficient than having each tool collect the same inventory information from each endpoint individually and leads to a more consistent understanding of enterprise state.

Software Inventory Message and Attributes for PA-TNC supports this activity through a number of mechanisms. As noted above, it allows a SW-PC to provide a complete list of software present in an endpoint's Software Inventory Evidence Collection to the SW-PV, which can then pass this information on to a central repository such as a Configuration Management Database (CMDB) or similar application. In addition, SW-PCs are required to be able to monitor for changes to an endpoint's Software Inventory Evidence Collection in near real-time and push reports of changes to the SW-PV as soon as those changes are detected. Thus any central repository fed by a SW-PV receiving such information can be updated soon after the change occurs. Keeping such a central repository synchronized with the state of each endpoint's Software Inventory Evidence Collection allows tools that use this information for their own security activities to make decisions in a consistent, efficient manner.

2.1.3. PA-TNC Use Cases

Software Inventory Message and Attributes for PA-TNC are intended to operate over the PA-TNC interface and, as such, are intended to meet the use cases set out in the PA-TNC specification.

2.2. Non-supported Use Cases

Some use cases not covered by this version of Software Inventory Message and Attributes for PA-TNC include:

- This specification does not address how the endpoint's Software Inventory Evidence Collection is populated. In particular, NEA components are not expected to perform software discovery activities beyond compiling information in an endpoint's Software Inventory Evidence Collection. This collection might potentially come from multiple sources on the endpoint (e.g., information generated dynamically by package management tools or discovery tools, as well as SWID tag files discovered on the file system). While an enterprise might make use of software discovery procedures to identify installed software such procedures are outside the scope of this specification.

- This specification does not address converting inventory information expressed in a proprietary format into formats used in the attributes described in this specification. Instead, it focuses exclusively on defining interfaces for the transportation of software information in the expectation that this is the format around which reporting tools will converge.

- This specification provides no mechanisms for a posture validator to request a specific list of software information based on arbitrary software properties. For example, requesting only information about software from a particular vendor is not supported. After the endpoint's software inventory evidence collection has been copied to some central location, such as the CMDB, processes there can perform queries based on any criteria present in the collected information, but this specification does not address using such queries to constrain the initial collection of this information from the endpoint.

- This specification does not address utilization of properties of certain sources of software information that might facilitate local tests (i.e., on the endpoint) of endpoint state. For example, the optional package_footprint field of an ISO SWID tag can contain a list of files and hash values associated with the software indicated by the tag. Tools on the endpoint can use the values in this field to test for the presence of the indicated

files. Successful evaluation of such tests leads to greater assurance that the indicated software is present on the endpoint. Currently, most SWID tag creators do not provide values for tag fields that support local testing. For this reason, the added complexity of supporting endpoint testing using these fields is out of scope for this specification. Future versions of this specification might add support for such testing.

2.3. Specification Requirements

Below are the requirements that the Software Inventory Message and Attributes for PA-TNC specification is required to meet in order to successfully play its role in the NEA architecture.

o Efficient

The NEA architecture enables delay of network access until the endpoint is determined not to pose a security threat to the network based on its asserted integrity information. To minimize user frustration, the Software Inventory Message and Attributes for PA-TNC ought to minimize overhead delays and make PA-TNC communications as rapid and efficient as possible.

Efficiency is also important when one considers that some network endpoints are small and low powered, some networks are low bandwidth and/or high latency, and some transport protocols (such as PT-EAP, Posture Transport (PT) Protocol for Extensible Authentication Protocol (EAP) Tunnel Methods [RFC7171]) or their underlying carrier protocol might allow only one packet in flight at a time or only one roundtrip. However, when the underlying PT protocol imposes fewer constraints on communications, this protocol ought to be capable of taking advantage of more robust communication channels (e.g. using larger messages or multiple roundtrips).

o Scalable

Software Inventory Message and Attributes for PA-TNC needs to be usable in enterprises that contain tens of thousands of endpoints or more. As such, it needs to allow a security tools to make decisions based on up-to-date information about an endpoint’s software inventory without creating an excessive burden on the enterprise’s network.

o Interoperable

This specification defines the protocol for how PCs and PVs can exchange and use software information to provide a NEA Server with information about an endpoint’s software inventory. Therefore a key

goal for this specification is ensuring that all SW PCs and PVs, regardless of the vendor who created them, are able to interoperate in their performance of these duties.

o Support precise and complete historical reporting

This specification outlines capabilities that support real-time understanding of the state of endpoint in a network in a way that can be used by other tools. One means of facilitating such an outcome is for a Configuration Management Database (CMDB) to be able to contain information about all endpoints connected to the enterprise for all points in time between the endpoint's first connection and the present. In such a scenario, it is necessary that any PC be able to report any changes to its software inventory evidence collection in near real-time while connected and, upon reconnection to the enterprise, be able to update the NEA Server (and through it the CMDB) with regard to the state of its software inventory evidence collection throughout the entire interval when it was not connected.

2.4. Non-Requirements

There are certain requirements that the Software Inventory Message and Attributes for PA-TNC specification explicitly is not required to meet. This list is not exhaustive.

o End to End Confidentiality

This specification does not define mechanism for confidentiality, nor is this property automatically provided by PA-TNC interface use. Confidentiality is generally provided by the underlying transport protocols, such as the PT Binding to TLS [RFC6876] or PT-EAP Posture Transport for Tunneled EAP Methods [RFC7171] - see Section 8 for more information on related standards. Should users wish confidentiality protection of assessment instructions or results, this needs to be provided by parts of the NEA architecture other than this specification.

2.5. Assumptions

Here are the assumptions that Software Inventory Message and Attributes for PA-TNC makes about other components in the NEA architecture.

o Reliable Message Delivery

The Posture Broker Client and Posture Broker Server are assumed to provide reliable delivery for PA-TNC messages and therefore the Attributes sent between the SW PCs and the PVs. In the event that

reliable delivery cannot be provided, the Posture Collector or Posture Validator is expected to terminate the connection.

2.6. Non-Assumptions

The Software Inventory Message and Attributes for PA-TNC specification explicitly does not assume:

- Authenticity and Accuracy of the Software Inventory Evidence Collection with Regard to Endpoint Inventory

This specification makes no assumption as to whether the software information that it reports correctly reflect the software state on the endpoint. This specification does not attempt to detect when the endpoint is providing false information, either through malice or error, but instead focuses on correctly and reliably providing the reported Software Inventory Evidence Collection to the NEA Server. Similarly, this specification makes no assumption with regard to the completeness of the Software Inventory Evidence Collection's coverage of the total set of software installed on the endpoint. It is possible, and even likely, that some installed software is not represented by a record in an endpoints Software Inventory Evidence Collection. See Section 6 for more on this security consideration.

2.7. Software Inventory Message and Attributes for PA-TNC Diagram Conventions

This specification defines the syntax of the Software Inventory Message and Attributes for PA-TNC using diagrams. Each diagram depicts the format and size of each field in bits. Implementations MUST send the bits in each diagram as they are shown from left to right for each 32-bit quantity traversing the diagram from top to bottom. Multi-octet fields representing numeric values MUST be sent in network (big endian) byte order.

Descriptions of bit fields (e.g. flags) values refer to the position of the bit within the field. These bit positions are numbered from the most significant bit through the least significant bit. As such, an octet with only bit 0 set would have a value of 0x80 (1000 0000), an octet with only bit 1 set would have a value of 0x40 (0100 0000), and an octet with only bit 7 set would have a value of 0x01 (0000 0001).

3. Software Inventory Message and Attributes for PA-TNC System Requirements

The Software Inventory Message and Attributes for PA-TNC specification facilitates the exchange of software inventory and event information. Specifically, each application supporting Software Inventory Message and Attributes for PA-TNC includes a component known as the SW-PC that receives messages sent with the SW Attributes component type. The SW-PC is also responsible for sending appropriate SW Attributes back to the SW-PV in response. This section outlines what software inventories and events are and the requirements on SW-PCs and SW-PVs in order to support the stated use cases of this specification.

3.1. Basic Attribute Exchange

In the most basic exchange supported by this specification, a SW-PV sends a request to the SW-PC requesting some type of information about the endpoint's software inventory. This simple exchange is shown in Figure 2.

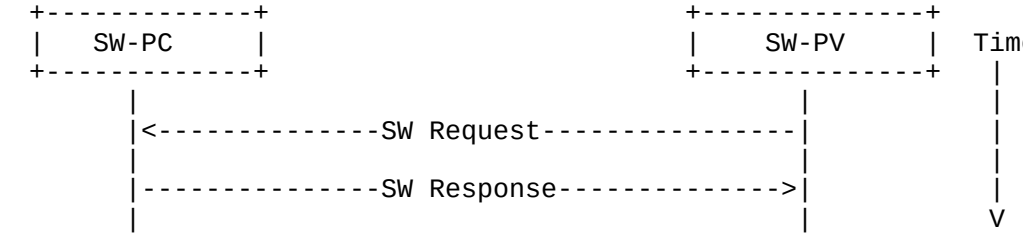


Figure 2: Basic SW Attribute Exchange

Upon receiving such a SW Request from the SW-PV, the SW-PC is expected to collect all the relevant software inventory information from the endpoint's software evidence collection and place it within its response attribute.

SW-PVs MUST discard without error any SW Response attributes that they receive for which they do not know the SW Request parameters that led to this SW Response. This is due to the fact that the SW Request includes parameters that control the nature of the response (as will be described in the following sections) and without knowing those parameters the SW Response cannot be reliably interpreted. Most often receiving an unsolicited SW Response attribute happens when a NEA Server has multiple SW-PVs; one SW-PV sends a SW Request but, unless exclusive delivery is used by the SW-PC in sending the

response, both SW-PVs receive copies of the resulting SW Response. In this case, the SW-PV that didn't send the SW Request would lack the context necessary to correctly interpret the SW Response it received and would simply discard it. Note, however, that proprietary measures might allow a SW-PV to discover the SW Request parameters for a SW Response even if that SW-PV did not send the given SW Request. As such, there is no blanket requirement for a SW-PV to discard all SW Responses to SW Request the SW-PV did not generate itself, only that SW-PVs are required to discard SW Responses for which they cannot get the necessary context to interpret.

In the case that it is possible to do so, the SW-PC SHOULD send its SW Response attribute to the SW-PV that requested it using exclusive delivery as described in section 4.5 of RFC 5793 (PB-TNC) [RFC5793]. Exclusive delivery ensures that only the sender of the SW Request receives the resulting SW Response.

3.2. Core Software Reporting Information

Different parameters in the SW Request can influence what information is returned in the SW Response. However, while each SW Response provides different additional information about this installed software, they all share a common set of fields that support reliable software identification on an endpoint. These fields include: Software Identifiers, Record Identifiers, and Software Locators. These three fields are present for each reported piece of software in each type of SW Response. The following sections examine these three types of information in more detail.

3.2.1. Software Identifiers

A Software Identifier uniquely identifies a specific version of a specific software product. The Software Inventory Message and Attributes for PA-TNC specification does not dictate the structure of a Software Identifier (beyond stating that it is a string) or define how it is created. Instead, each data model described in the Software Data Model IANA table (Section 9.4) includes its own rules for how a Software Identifier is created based on a record in the Endpoint's Software Inventory Evidence Collection expressed in that data model. Other data models will have their own procedures for the creation of associated Software Identifiers. Within the Software Inventory Message and Attributes for PA-TNC specification, the Software Identifier is simply an opaque string and there is never any need to unpack any information that might be part of that identifier.

A Software Identifier is a fraction of the size of the inventory record from which it is derived. For some combinations of data

models and sources, the full record might never be necessary as the identifier can be directly correlated to the contents of the full record. This is possible with authoritative SWID tags, since these tags always have the same contents and thus a Software Identifier derived from these tags can be used as a lookup to a local copy of the full tag. For other combinations of source and data model, a server might not be able to determine the specific software product and version associated with the identifier without requesting delivery of the full record. However, even in those cases, downstream consumers of this information might never need the full record as long as the Software Identifiers they receive can be tracked reliably. A SW-PV can use Software Identifiers to track the presence of specific software products on an endpoint over time in a bandwidth-efficient manner.

There are two important limitations of Software Identifiers to keep in mind:

1. The identifiers do not necessarily change when the associated record changes. In some situations, a record in the endpoint's Software Inventory Evidence Collection will change due to new information becoming available or in order to correct prior errors in that information. Such changes might or might not result in changes to the Software Identifier, depending on the nature of the changes and the rules governing how Software Identifiers are derived from records of the appropriate data model.

2. It is possible that a single software product is installed on a single endpoint multiple times. If both of these installation instances are reported by the same source using the same data format, then this can result in identical Software Identifiers for each installation instances. In other words, Software Identifiers might not uniquely identify installation instances; they just are intended to uniquely identify software products (which might have more than one installation instance). Instead, to reliably distinguish between multiple instances of a single software product, one needs to make use of Record Identifiers, described in the following section.

3.2.1.1. Record Identifiers

A Record Identifier is a 4-byte integer generated by the SW-PC that is uniquely associated with a specific record within the Endpoint's Software Inventory Evidence Collection. The SW-PC MUST assign a unique identifier to each record when it is added to the Endpoint's Software Inventory Evidence Collection. The Record Identifier SHOULD remain unchanged if that record is modified. The SWID-PC might wish

to assign Record Identifiers sequentially, but any scheme is acceptable provided that no two records receive the same identifier.

Servers can use Record Identifiers to distinguish between multiple instances of a single software product installed on an endpoint. Since each installation instance of a software product is associated with a separate record, servers can use the record identifier to distinguish between instances. For example, if an event is reported (as described in Section 3.5), the record identifier will allow the server to discern which instance of a software product is involved.

3.2.1.2. Software Locators

In addition to the need to identify software products, many use cases of inventory information need to know where software is located on the endpoint. This information might be needed to direct remediation actions or similar processes. For this reason, every reported software product also includes a Software Locator to identify where the software is installed on the endpoint.

If the location is not provided directly by the record source the SW-PC is responsible for attempting to determine the location of the software product. The "location" of a product SHOULD be the directory in which the software products executables are kept. However, if that directory is shared by other software products, the "location" SHOULD be the location of the primary executable associated with the software product. The source and/or SW-PC MUST be consistent in reporting the location of a software product (i.e., it cannot use the executable location in one report and the directory location in another).

The location is expressed as a URI string consisting of a scheme and path. [RFC3986] The location URI does not include an authority part. The URI schema describes the context of the described location. For example, in most cases the location of the installed software product will be expressed in terms of its path in the filesystem. For such locations, the location URI scheme MUST be "file". It is possible that other schemes could be used to represent other location contexts. Apart from reserving the "file" and "unknown" (described below) scheme to indicate an installation location expressed using a path in the endpoint’s file system, this specification does not reserve schemes. When representing software products in other location contexts, tools MUST be consistent in their use of schemes, but the exact string used in those schemes is not normatively defined here.

It is possible, that a SW-PC is unable to determine the location of a reported software product. In this case, the SW-PC MUST assign that

software product a location of "unknown:". (I.e., the "unknown" scheme and an empty path.) However, SW-PCs SHOULD only make such an assignment as a last resort. Even a probable location for a software product is preferable to using the unknown indicator.

3.2.1.3. Using Software and Record Identifiers in SW Attributes

A SW Attribute reporting an endpoint's Software Inventory Evidence Collection always contains the Software Identifiers associated with the identified software products. A SW Attribute might or might not also contain copies of software inventory evidence records. The attribute exchange is identical to the diagram shown in Figure 2 regardless of whether software inventory evidence records are included. The SW Request attribute indicates whether the response is required to include software inventory evidence records. Excluding software inventory evidence records can reduce the attribute size of the response by multiple orders of magnitude when compared to sending the same inventory with full records.

3.3. Targeted Requests

Sometimes a SW-PV does not require information about every piece of software on an endpoint but only needs to receive updates about certain software instances. For example, enterprise endpoints might be required to have certain software products installed and to keep these updated. Instead of requesting a complete inventory just to see if these products are present, the SW-PV can make a "targeted request" for the software in question.

Targeted requests follow the same attribute exchange described in Figure 2. The SW-PV targets its request by providing one or more Software Identifiers in its SW Request attribute. The SW-PC MUST then limit its response to contain only records that match the indicated Software Identifier(s). This allows the network exchange to exclude information that is not relevant to a given policy question, thus reducing unnecessary bandwidth consumption. The SW-PC's response might or might not include software inventory evidence records, depending on the parameters of the SW Request.

Note that targeted requests identify the software relevant to the request only through Software Identifiers. This specification does not support arbitrary, parameterized querying of records. For example, one cannot request all records from a certain software publisher, or all records created by a particular record source. Targeted requests only allow a requestor to request specific software (as identified by their Software Identifiers) and receive a response that is limited to the named software.

There is no assumption that a SW-PC will recognize "synonymous records" - that is, records from different sources for the same software. Recall that different sources and data models may use different Software Identifier strings for the same software product. The SW-PC returns only records that match the Software Identifiers in the SW Request, even if there might be other records in the endpoint's Software Inventory Evidence Collection for the same software product. This is necessary because SW-PCs might not have the ability to determine that two Software Identifiers refer to the same product.

Targeted requests do not include Record Identifiers or Software Locators. The response to a targeted request MUST include all records associated with the named Software Identifiers, including the case where there are multiple records associated with a single Software Identifier.

SW-PCs MUST accept targeted requests and process them correctly as described above. SW-PVs MUST be capable of making targeted requests and processing the responses thereto.

3.4. Monitoring Changes in an Endpoint's Software Inventory Evidence Collection

The software collection on an endpoint is not static. As software is installed, uninstalled, patched, or updated, the Software Inventory Evidence Collection is expected to change to reflect the new software state on the endpoint. Different record sources might update the evidence collection at different rates. For example, a package manager might update its records in the Software Inventory Evidence Collection immediately whenever it is used to add or remove a software product. By contrast, sources that perform periodic examination of the endpoint would likely only update their records in the Software Inventory Evidence Collection after each examination.

All SW-PCs MUST be able to be able to detect changes to the Software Inventory Evidence Collection. Specifically, SW-PCs MUST be able to detect:

- The creation of records
- The deletion of records
- The alteration of records

An "alteration" is anything that modifies the contents of a record (or would modify it, if the record is dynamically generated on

demand) in any way, regardless of whether the change is functionally meaningful.

SW-PCs MUST detect such changes to the endpoint's Software Inventory Evidence Collection in close to real-time (i.e., within seconds) when the Posture Collector is operating. In addition, in the case where there is a period during which the SW-PC is not operating, the SW-PC MUST be able to determine the net change to the endpoint's Software Inventory Evidence Collection over the period it was not operational. Specifically, the "net change" represents the difference between the state of the endpoint's Software Inventory Evidence Collection when the SW-PC was last operational and monitoring its state, and the state of the endpoint's software inventory evidence collection when the SW-PC resumed operation. Note that a net change might not reflect the total number of change events over this interval. For example, if a record was altered three times during a period when the SW-PC was unable to monitor for changes, the net change of this interval might only note that there was an alteration to the record, but not how many individual alteration events occurred. It is sufficient for a SW-PC's determination of a net change to note that there was a difference between the earlier and current state rather than enumerating all the individual events that allowed the current state to be reached.

The SW-PC MUST assign a time to each detected change in the endpoint's Software Inventory Evidence Collection. These timestamps correspond to the SW-PC's best understanding as to when the detected change occurred. These timestamps MUST be as accurate as possible. For changes to the endpoint's Software Inventory Evidence Collection that occur while the SW-PC is operating, the SW-PC ought to be able to assign a time to the event that is accurate to within a few seconds. For changes to the endpoint's Software Inventory Evidence Collection that occur while the SW-PC is not operational, upon becoming operational the SW-PC needs to make a best guess as to the time of the relevant events (possibly by looking at timestamps on files), but these values might be off. In the case of dynamically generated records, the time of change is the time at which the data from which the records are generate changes, not the time at which a changed record is generated. For example, if records are dynamically generated based on data in an RPM database, the time of change would be when the RPM database changed.

With regard to deletions of records, the SW-PC needs to detect the deletion and MUST retain a copy of the full deleted record along with the associated Record Identifier and Software Locator so that the record and associated information can be provided to the SW-PV upon request. This copy of the record MUST be retained for a reasonable amount of time. Vendors and administrators determine what

"reasonable" means, but a copy of the record SHOULD be retained for as long as the event recording the deletion of the record remains in the SW-PC's event log (as described in Section 3.5). This is recommended because, as long as the event is in the SW-PC's change logs, the SW-PC might send an event attribute (described in Section 3.5) that references this record, and a copy of the record is needed if the SW-PV wanted a full copy of the relevant records.

With regard to alterations to a record, SW-PCs MUST detect any alterations to the contents of a record. Alterations need to be detected even if they have no functional impact on the record. A good guideline is that any alteration to a record that might change the value of a hash taken on the record's contents needs to be detected by the SW-PC. A SW-PC might be unable to distinguish modifications to the content of a record from modifications to the metadata the file system associates with the record. For example, a SW-PC might use the "last modification" timestamp as an indication of alteration to a given record, but a record's last modification time can change for reasons other than modifications to the record contents. A SW-PC is still considered compliant with this specification if it also reports metadata change events that do not change the record itself as alterations to the record. In other words, while SW-PC authors are encouraged to exclude modifications that do not affect the bytes within the record, discriminating between modifications to file contents and changes to file metadata can be difficult and time consuming on some systems. As such, as long as the alterations detected by a SW-PC always cover all modifications to the contents of record, the SW-PC is considered compliant even if it also registers alterations that do not modify the contents of a record as well. When recording an alteration to a record, the SW-PC is only required to note that an alteration occurred. The SW-PC is not required to note or record how the record altered, nor is it possible to include such details in SW Attributes reporting the change to a SW-PV. There is no need to retain a copy of the original record.

When a record changes it SHOULD retain the same Record Identifier. The Software Locator might or might not change, depending on whether the software changed locations during the changes that led to the record change. A record change MUST retain the same Software Identifier. This means that any action that changes a software product (e.g., application of a patch that results in a change to the product's version) MUST NOT be reflected by a record change but instead MUST result in the deletion of the old record and the creation of a new record. This reflects the requirement that a record in the endpoint's Software Inventory Evidence Collection correspond directly with an instance of a specific software product.

3.5. Reporting Change Events

As noted in the preceding section, SW-PCs MUST be able to detect changes to the endpoints Software Inventory Evidence Collection (creation, deletion, and alteration) in near real-time while the SW-PC is operational, and MUST be able to account for any net change to the endpoint's Software Inventory Evidence Collection that occurs when the SW-PC is not operational. However, to be of use to the enterprise, the NEA Server needs to be able to receive these events and be able to understand how new changes relate to earlier changes. In Software Inventory Message and Attributes for PA-TNC, this is facilitated by reporting change events. All SW-PCs MUST be capable of receiving requests for change events and sending change event attributes. All SW-PVs MUST be capable of requesting and receiving change event attributes.

3.5.1. Event Identifiers

To be useful, change events need to be correctly ordered. Ordering of events is facilitated by two pieces of information: an Event Identifier (EID) value and an EID Epoch value.

An EID is a 4-byte unsigned integer that the SW-PC assigns sequentially to each observed event (whether detected in real-time or deduced by looking for net changes over a period of SW-PC inactivity). All EIDs exist within the context of some "EID Epoch", which is also represented as a 4-byte unsigned integer. EID Epochs are used to ensure synchronization between the SW-PC and any SW-PVs with which it communicates. EID Epoch values SHOULD be generated randomly and in such a way that it is unlikely that the same EID Epoch is generated twice, even if the SW-PC reverted to an earlier state (e.g., resetting it to factory defaults). In the case where a SW-PC needs to reset its EID counter, either because it has exhausted all available EID values or because the SW-PC's event log becomes corrupted, then a new EID Epoch MUST be selected.

Within an Epoch, EIDs MUST be assigned sequentially, so that if a particular event is assigned an EID of N, the next observed event is given an EID of N+1. In some cases, events might occur simultaneously, or the SW-PC might not otherwise be able to determine an ordering for events. In these cases, the SW-PC creates an arbitrary ordering of the events and assigns EIDs according to this ordering. Two change events MUST NOT ever be assigned the same EID within the same EID Epoch. No meaningful comparison can be made between EID values of different Epochs.

The EID value of 0 is reserved and MUST NOT be associated with any event. Specifically, an EID of 0 in a SW Request attribute indicates

that a SW-PV wants an inventory response rather than an event response, while an EID of 0 in a SW Response is used to indicate the initial state of the endpoint's Software Inventory Evidence Collection prior to the observation of any events. Thus the very first recorded event in a SW-PC's records within an EID Epoch MUST be assigned a value of 1 or greater. Note that EID and EID Epoch values are assigned by the SW-PC without regard to whether events are being reported to one or more SW-PVs. The SW-PC records events and assigns EIDs during its operation. Any and all SW-PVs that request event information from the SW-PC will have those requests served from the same event records and thus will see the same EIDs and EID Epochs for the same events.

The SW-PC MUST ensure there is no coverage gap (i.e., change events that are not recorded in the SW-PC's records) in its change event records. This is necessary because a coverage gap might give a SW-PV a false impression of the endpoint's state. For example, if a SW-PV saw an event indicating that a particular record had been added to the endpoint's software inventory evidence collection, and saw no subsequent events indicating that record had been deleted, it might reasonably assume that this record was still present and thus that the indicated software was still installed (assuming the Epoch has not changed). If there is a coverage gap in the SW-PC's event records, however, this assumption could be false. For this reason, the SW-PC's event records MUST NOT contain gaps. In the case where there are periods where it is possible that changes occurred without the SW-PC detecting or recording them, the SW-PC MUST either compute a net change and update its event records appropriately, or pick a new EID Epoch to indicate a discontinuity with previous event records.

Within a given Epoch, once a particular event has been assigned an EID, this association MUST NOT be changed. That is, within an Epoch, once an EID is assigned to an event, that EID cannot be reassigned to a different event, and the event cannot be assigned a different EID. When the SW-PC's Epoch changes, all of these associations between EIDs and events are cancelled, and EID values once again become free for assignment.

3.5.2. Core Event Tracking Information

Whether reporting events or full inventories it is important to know how the reported information fits into the overall timeline of change events. This is why all SW Response attributes include fields to place that response within the sequence of detected events. Specifically, all SW Responses include a Last EID and EID Epoch field. The EID Epoch field identifies the EID Epoch in which the SW Response was sent. If the SW Response is reporting events, all

reported events occurred within the named EID Epoch. The Last EID (which is also always from the named EID Epoch) indicates the EID of the last recorded change event at the time that the SW Response was sent. These two fields allow any response to be placed in the context of the complete set of detected change events within a given EID Epoch.

3.5.3. Updating Inventory Knowledge Based on Events

Modern endpoints can have hundreds of software products installed, most of which are unlikely to change from one day to the next. As such, instead of exchanging a complete list of an endpoint's inventory on a regular basis, one might wish to only identify changes since some earlier known state of this inventory. This is readily facilitated by the use of EIDs to place change events in a context relative to earlier state.

As noted above, every SW Response sent by a SW-PC to a SW-PV (as described in Section 3.1 through Section 3.3) includes the EID Epoch and EID of the last event recorded prior to that response being compiled. This allows the SW-PV to place all subsequently received event records in context relative to this SW Response attribute (since the EIDs represent a total ordering of all changes to the endpoint's software inventory evidence collection). Specifically, a SW-PV (or, more likely, a database that collects and records its findings) can record an endpoint's full inventory and also the EID and Epoch of the most recent event reflected at the time of that inventory. From that point on, if change events are observed, the attribute describing these events indicates the nature of the change, the affected records, and the order in which these events occurred (as indicated by the sequential EIDs). Using this information, any remote record of the endpoint's Software Inventory Evidence Collection can be updated appropriately.

3.5.4. Using Event Records in SW Attributes

A SW-PV MUST be able to request a list of event records instead of an inventory. The attribute flow in such an exchange looks the same as the basic flow shown in Figure 2. The only difference is that, in the SW Request attribute, the SW-PV provides an EID other than 0. (A value of 0 in these fields represents a request for an inventory.) When the SW-PC receives such a request, instead of identifying records from the endpoint's Software Inventory Evidence Collection, it consults its list of detected changes. The SW-PC MUST add an event record to the SW Response attribute for each recorded change event with an EID greater than or equal to the EID in the SW Request attribute (although targeting of requests, as described in the next

paragraph, might limit this list). A list of event records MUST only contain events with EIDs that all come from the current Epoch.

SW-PVs can target requests for event records by including one or more Software Identifiers, as described in Section 3.3, in the SW Request that requests an event record list. A targeted request for event records is used to indicate that only events affecting software that matches one of the provided Software Identifiers are to be returned. Specifically, in response to a targeted request for event records, the SW-PC MUST exclude any event records that are less than the indicated EID (within the current EID Epoch) and exclude any event records where the affected software does not match one of the provided Software Identifiers. This might mean that the resulting list of event records sent in the response attribute does not provide a continuous sequence of EIDs. Both SW-PCs and SWIC-PVs MUST support targeted request for event records.

3.5.5. Partial and Complete Lists of Event Records in SW Attributes

Over time, a SW-PC might record a large number of change events. If a SW-PV requests all change events covering a large period of time, the resulting SW Response attribute might be extremely large, especially if the SW-PV requests inclusion of software inventory evidence records in the response. In the case that the resulting attribute is too large to send (either because it exceeds the 4GB attribute size limit imposed by the PA-TNC specification, or because it exceeds some smaller size limit imposed on the SW-PC) the SW-PC MAY send a partial list of event records back to the SW-PV.

Generation of a partial list of events in a SW Response attribute requires the SW-PC to identify a "consulted range" of EIDs. A consulted range is the set of event records that are examined for inclusion in the SW Response attribute and that are included in that attribute if applicable. Recall that, if a SW Request is targeted, only event records that involve the indicated software would be applicable. (See Section 3.3 for more on Targeted Request.) If a request is not targeted, all event records in the considered range are applicable and included in the SW Response attribute.

The lower bound of the consulted range MUST be the EID provided in the SW Request. (Recall that a SW Request indicates a request for event records by providing a non-0 EID value in the SW Request. See Section 3.5.4.) The upper bound of the consulted range is the EID of the latest event record (as ordered by EID values) that is included in the SW Response attribute if it is applicable to the request. The EID of this last event record is called the "Last Consulted EID". The SW-PC chooses this Last Consulted EID based on the size of the event record list it is willing to provide to the SW-PV.

A partial result list MUST include all applicable event records within the consulted range. This means that for any applicable event record (i.e., any event record in an un-targeted request, or any event record associated with software matching a requested Software Identifier in a targeted request) whose EID is greater than or equal to the EID provided in the SW Request and whose EID is less than or equal to the Last Consulted EID, that event record MUST be included in the SW Response conveying this partial list of event records. This ensures that every partial list of event records is always complete within its indicated range.

All SW Response attributes that convey event records include a Last Consulted EID field. This is in addition to the EID Epoch and Last EID fields that are present in all SW Responses. Note that, if responding to a targeted SW Request, the SW Response attribute might not contain the event record whose EID matches the Last Consulted EID value. For example, the last consulted EID record might have been deemed inapplicable because it did not match the specified list of Software Identifiers in the SW Request.

If a SW-PV receives a SW Response attribute where the Last EID and Last Consulted EID fields are identical, the SW-PV knows that it has received a result list that is complete, given the parameters of the request, up to the present time.

On the other hand, if the Last EID is greater than the Last Consulted EID, the SW-PV has received a partial result list. (The Last Consulted EID MUST NOT exceed the Last EID.) In this case, if the SW-PV wishes to try to collect the rest of the partially delivered result list it then sends a new SW Request whose EID is one greater than the Last Consulted EID in the preceding response. Doing this causes the SW-PC to generate another SW Response attribute containing event records where the earliest reported event record is the one immediately after the event record with the Last Consulted EID (since EIDs are assigned sequentially). By repeating this process until it receives a SW Response where the Last EID and Last Consulted EID are equal, the SW-PV is able to collect all event records over a given range, even if the complete set of event records would be too large to deliver via a single attribute.

Implementers need to be aware that a SW Request might specify an EID that is greater than the EID of the last event recorded by a SW-PC. In accordance with the behaviors described in Section 3.5.4, a SW-PC MUST respond to such a request with a SW Response attribute that contains zero event records. This is because the SW-PC has recorded no event records with EIDs greater than or equal to the EID in the SW Request. In such a case, the Last Consulted EID field MUST be set to the same value as the Last EID field in this SW Response attribute.

This case is called out because the consulted range on a SW-PC in such a situation is a negative range, where the "first" EID in the range (provided in the SW Request) is greater than the "last" EID in the range (this being the EID of the last recorded event on the SW-PC). Implementers need to ensure that SW-PCs do not experience problems in such a circumstance.

Note that this specification only supports the returning of partial results when returning event records. There is no way to return a partial inventory list under this specification.

3.5.6. Synchronizing Event Identifiers and Epochs

Since EIDs are sequential within an Epoch, if a SW-PV's list of event records contains gaps in the EID values within a single Epoch, the SW-PV knows that there are events that have not been accounted for. The SW-PV can either request a new event list to collect the missing events or request a full inventory to re-sync its understanding of the state of the endpoint's Software Inventory Evidence Collection. In either case, after the SW-PV's record of the endpoint's Software Inventory Evidence Collection has been updated, the SW-PV can record the new latest EID value and track events normally from that point on.

If the SW-PV receives any attribute from a SW-PC where the EID Epoch differs from the EID Epoch that was used previously, then SW-PV or any entity using this information to track the endpoint's Software Inventory Evidence Collection knows that there is a discontinuity in their understanding of the endpoint's state. To move past this discontinuity and reestablish a current understanding of the state of the endpoint's Software Inventory Evidence Collection, the SW-PV needs to receive a full inventory from the endpoint. The SW-PV cannot be brought in sync with the endpoint's state through the collection of any set of event records in this situation. This is because it is not possible to account for all events on the SW-PC since the previous Epoch was used, because there is no way to query for EIDs from a previous Epoch. Once the SW-PV has received a full inventory for the new Epoch, the SW-PV records the latest EID reported in this new Epoch and can track further events normally.

A SW-PC MUST NOT report events with EIDs from any Epoch other than the current EID Epoch. The SW-PC MAY choose to purge all event records from a previous Epoch from memory after an Epoch change. Alternately, the SW-PC MAY choose to retain some event records from a previous EID Epoch and assign them new EIDs in the current Epoch. However, in the case where a SW-PC chooses the latter option it MUST ensure that the order of events according to their EIDs is unchanged and that there is no coverage gap between the first retained event

recorded during the previous Epoch (now reassigned with an EID in the current Epoch) and the first event recorded during the current Epoch. In particular, the SW-PC MUST ensure that all change events that occurred after the last recorded event from the previous Epoch are known and recorded. (This might not be possible if the Epoch change is due to state corruption on the SW-PC.) A SW-PC might choose to reassign EIDs to records from a preceding Epoch to create a "sliding window" of events, where each Epoch change represents a shift in the window of available events.

In the case where a SW-PC suffers a crash and loses track of its current EID Epoch or current EID, then it MUST generate a new EID Epoch value and begin assigning EIDs within that Epoch. In this case, the SW-PC MUST purge all event records from before the crash as it cannot ensure that there is not a gap between the last of those records and the next detected event. The process for generating a new EID Epoch MUST minimize the possibility that the newly generated EID Epoch is the same as a previously used EID Epoch.

The SW-PV will normally never receive an attribute indicating that the latest EID is less than the latest EID reported in a previous attribute within the same EID Epoch. If this occurs, the SW-PC has suffered an error of some kind, possibly indicative of at least partial corruption of its event log. In this case, the SW-PV SHOULD treat the situation as if there was a change in Epoch and treat any local copy of the endpoint’s Software Inventory Evidence Collection as out-of-sync until a full inventory can be reported by the SW-PC. In this case, the SW-PV SHOULD flag the occurrence so the SW-PC can be examined to ensure it is now operating properly.

3.6. Subscriptions

Thus far, all attribute exchanges discussed assume that a SW-PV sent an SW Request attribute and the SW-PC is providing a direct response to that request. The Software Inventory Message and Attributes for PA-TNC specification also supports the ability for a SW-PC to send a SW Response to the SW-PV in response to observed changes in the endpoint’s software inventory evidence collection, instead of in direct response to a SW Request. An agreement by a SW-PC to send content when certain changes are detected to the endpoint’s Software Inventory Evidence Collection is referred to in this specification as a "subscription", and the SW-PV that receives this content is said to be "subscribed to" the given SW-PC. All SW-PCs and SW-PVs MUST support the use of subscriptions.

3.6.1. Establishing Subscriptions

A SW-PV establishes a subscription on a particular SW-PC by sending a SW Request attribute with the Subscription flag set. The SW Request attribute is otherwise identical to the SW Requests discussed in previous sections. Specifically, such a SW Request might or might not request inclusion of software inventory evidence records, might or might not be targeted, and might request change event records or endpoint inventory. Assuming no error is encountered, a SW-PC MUST send a SW Response attribute in direct response to this SW Request attribute, just as if the Subscription flag was not set. As such, the attribute exchange that establishes a new subscription in a SW-PC has the same flow seen in the previous attribute exchanges, as depicted in Figure 2. If the SW-PV does not receive a PA-TNC Error attribute (as described in Section 3.8 and Section 4.14) in response to their subscription request, the subscription has been successfully established on the SW-PC. The SW Request attribute that establishes a new subscription is referred to as the "establishing request" for that subscription.

When a subscription is established it is assigned a Subscription ID value. The Subscription ID is equal to the value of the Request ID of the establishing request. (For more about Request IDs, see Section 4.6.)

A SW-PC MUST have the ability to record and support multiple simultaneous subscriptions from a single party and from multiple parties. A SW-PV MUST have the ability to record and support multiple simultaneous subscriptions to a single party and subscriptions to multiple parties.

3.6.2. Managing Subscriptions

The SW-PC MUST record each accepted subscription along with the identity of the party to whom attributes are to be pushed in compliance with the subscription. This identity includes both the NEA Server’s connection ID and the Posture Validator Identifier from the PB-PA message that delivered the request.

Likewise, SW-PVs MUST record each accepted subscription for which they are the subscribing party along with the associated Subscription ID and the identity of the SW-PC that will be fulfilling the subscription. The SW-PV needs to retain this information in order to correctly interpret pushed SW Response attributes sent in fulfillment of the subscription. The identity of the SW-PC is given in the Posture Collector Identifier of the PB-PA message header in all messages from that SW-PC.

3.6.3. Terminating Subscriptions

Subscriptions MAY be terminated at any time by the subscribing SW-PV by setting the Clear Subscriptions flag in a SW Request. (See Section 4.7 for more on using this flag.) In the case that a SW Request with the Clear Subscriptions flag set is received the SW-PC MUST only clear subscriptions that match both the NEA server connection ID and the SW-PV ID for this SW Request, and MUST clear all such subscriptions.

This specification does not give the SW-PV the ability to terminate subscriptions individually - all subscriptions to the SW-PV are cleared when the Clear Subscriptions flag is set.

This specification does not give the SW-PC the ability to unilaterally terminate a subscription. However, if the SW-PC experiences a fatal error fulfilling a subscription, resulting in sending a PA-TNC Error attribute of type SW_SUBSCRIPTION_FULFILLMENT_ERROR, then the subscription whose fulfillment led to the error MUST be treated as terminated by both the SW-PC and the SW-PV. Only the subscription experiencing the error is cancelled and other subscriptions are unaffected. See Section 3.8 for more on this error condition.

Finally, a subscription is terminated if the connection between the SW-PC and SW-PV is deleted. This occurs when the connection ID used in the messages between the SW-PC and the SW-PV becomes unbound. Loss of this connection ID would prevent the SW-PC from sending messages in fulfillment of this subscription. As such, loss of the connection ID necessarily forces subscription termination between the affected parties.

3.6.4. Subscription Status

A SW-PV can request that a SW-PC report the list of active subscriptions for which the SW-PV is the subscriber. A SW-PV can use this to recover lost information about active subscriptions. A SW-PV can also use this capability to verify that a SW-PC has not forgotten any of its subscriptions. The latter is especially useful where a SW-PC does not send any attributes in fulfillment of a given subscription for a long period of time. The SW-PV can check the list of active subscriptions on the SW-PC and verify whether the inactivity is due to a lack of reportable events or due to the SW-PC forgetting its obligations to fulfill a given subscription.

A SW-PV requests a list of its subscriptions on a given SW-PC by sending that SW-PC a Subscription Status Request. The SW-PC MUST then respond with a Subscription Status Response (or a PA-TNC Error

if an error condition is experienced). The Subscription Status Response MUST contain one subscription record for each of the active subscriptions for which the SW-PV is the subscribing party.

3.6.5. Fulfilling Subscriptions

As noted in Section 3.4 SW-PCs MUST have the ability to automatically detect changes to an endpoint's Software Inventory Evidence Collection in near real-time. For every active subscription, the SW-PC MUST send an attribute to the subscribed SW-PV whenever a change is detected to relevant records within the endpoint's Software Inventory Evidence Collection. Such an attribute is said to be sent "in fulfillment of" the given subscription and any such attribute MUST include that subscription's Subscription ID. If the establishing request for that subscription was a targeted request, then only records that match the Software Identifiers provided in that establishing request are considered relevant. Otherwise, (i.e., for non-targeted requests) any record is considered relevant for this purpose. Figure 3 shows a sample attribute exchange where a subscription is established and then later attributes are sent from the SW-PC in fulfillment of the established subscription.

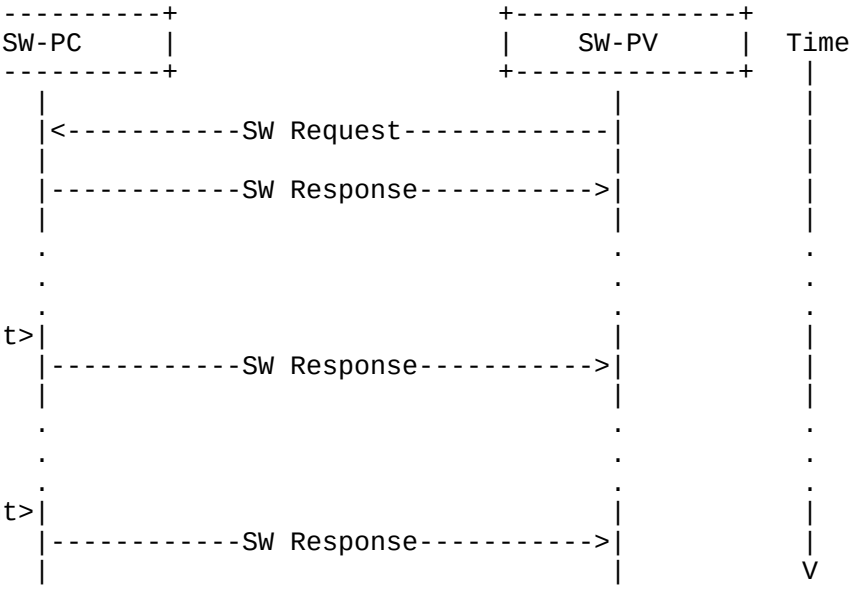


Figure 3: Subscription Establishment and Fulfillment

The contents of an attribute sent in fulfillment of a subscription depend on the parameters provided in the establishing request for that subscription. Specifically, the contents of an attribute sent in fulfillment of a subscription have the same format as would a direct response to the establishing request. For example, if the establishing request stipulated a response that contained an event record list that included software inventory evidence records, all attributes sent in fulfillment of this subscription will also consist of event record lists with software inventory evidence records. As such, all SW Responses displayed in the exchange depicted in Figure 3 have the same format. A SW Response generated in fulfillment of an active subscription MUST be a valid SW Response attribute according to all the rules outlined in the preceding sections. In other words, an attribute constructed in fulfillment of a subscription will look the same as an attribute sent in direct response to an explicit request from a SW-PV that had the same request parameters and which arrived immediately after the given change event. There are a few special rules that expand on this guideline:

3.6.5.1. Subscriptions Reporting Inventories

In the case that a SW-PV subscribes to a SW-PC requesting an inventory attribute whenever changes are detected (i.e. the EID in the establishing request is 0), then the SW-PC MUST send the requested inventory whenever a relevant change is detected. (A "relevant change" is any change for untargeted requests, or a change to an indicated record in a targeted request.) Upon detection of a relevant change for an active subscription, the SW-PC sends the appropriate inventory information as if it had just received the establishing request. Attributes sent in fulfillment of this subscription will probably have a large amount of redundancy, as the same records are likely to be present in each of these SW Attributes. The role of an inventory subscription is not to report records just for the items that changed - that is the role of a subscription that reports events (see Section 3.6.5.2). A SW-PC MUST NOT exclude a record from an attribute sent in fulfillment of an inventory subscription simply because that record was not involved in the triggering event (although a record might be excluded for other reasons, such as if the subscription is targeted - see Section 3.6.5.3).

3.6.5.2. Subscriptions Reporting Events

The way in which a SW-PV indicates it wishes to establish a subscription requesting event records is by providing a non-zero EID in the SW Request establishing the subscription (see Section 3.5.1). However, when the SW-PC constructs an attribute in fulfillment of the subscription (other than the direct response to the establishing

request), it MUST only include event records for the detected change(s) that precipitated this response attribute. In other words, it MUST NOT send a complete list of all changes starting with the indicated EID, up through the latest change, every time a new event is detected. In effect, the EID in the establishing request is treated as being updated every time an attribute is sent in fulfillment of this subscription, such that a single event is not reported twice in fulfillment of a single subscription. As such, every SW-PC MUST track the EID of the last event that triggered an attribute for the given subscription. When the next event (or set of events) is detected, the SW-PC MUST only report events with EIDs after the last reported event. In the case that the EID Epoch of the SW-PC changes, the SW-PC MUST treat EID values in the new Epoch as being after all EIDs assigned in the previous Epoch regardless of the relative numeric values of these EIDs.

Note that while a subscription is active, the subscribing SW-PV MAY make other requests for event records that overlap with events that are reported in fulfillment of a subscription. Such requests are unaffected by the presence of the subscription, nor is the subscription affected by such requests. In other words, a given request will get the same results back whether or not there was a subscription. Likewise, an attribute sent in fulfillment of a subscription will contain the same information whether or not other requests had been received from the SW-PV.

A SW-PV needs to pay attention to the EID Epoch in these attributes, as changes in the Epoch might create discontinuities in the SW-PV’s understanding of the endpoint’s Software Inventory Evidence Collection state, as discussed in Section 3.5.6. In particular, once the EID Epoch changes, a SW-PV is unable have confidence that it has a correct understanding of the state of an endpoint’s Software Inventory Evidence Collection until after the SW-PV collects a complete inventory.

SW-PCs MAY send partial lists of event records in fulfillment of a subscription. (See Section 3.5.5 for more on partial list of event records.) In the case that a SW-PC sends a partial list of event records in fulfillment of a subscription, it MUST immediately send the next consecutive partial list, and continue doing so until it has sent the equivalent of the complete list of event records. In other words, if the SW-PC sends a partial list it does not wait for another change event to send another SW Response, but continues sending SW Responses until it has sent all event records that would have been included in a complete fulfillment of the subscription. Note that the direct response to the establishing request is not considered to be sent in fulfillment of a subscription. However, in this case the SW-PC MUST treat the presence of unreported events as a triggering

event for pushing additional messages in fulfillment of the newly established subscription. As such, the net effect is that, if the direct response to the establishing request (i.e., the Subscription Fulfillment flag is unset) is partial, the SW-PC will immediately follow this with additional attributes (with the Subscription Fulfillment flag set) until the complete set of events has been sent to the SW-PV.

3.6.5.3. Targeted Subscriptions

Subscriptions MAY be targeted to only apply to records that match a given set of Software Identifiers. In the case where changes are detected that affect multiple records, some matching the establishing request's Software Identifiers and some not, the attribute sent in fulfillment of the subscription MUST only include inventory or events (as appropriate) for records that match the establishing request's Software Identifiers. The SW-PC MUST NOT include non-matching records in the attribute, even if those non-matching records experienced change events that were co-temporal with change events on the matching records.

In addition, a SW-PC MUST send an attribute in fulfillment of a targeted subscription only when changes to the endpoint's Software Inventory Evidence Collection impact one or more records matching the subscription's establishing request's Software Identifiers. A SW-PC MUST NOT send any attribute in fulfillment of a targeted subscription based on detected change to the endpoint's Software Inventory Evidence Collection that did not involve any of the records targeted by that subscription.

3.6.5.4. No Subscription Consolidation

A SW-PV MAY establish multiple subscriptions to a given SW-PC. If this is the case, it is possible that a single change event on the endpoint might require fulfillment by multiple subscriptions, and that the information included in attributes that fulfill each of these subscriptions might overlap. The SW-PC MUST send separate attributes for each established subscription that requires a response due to the given event. Each of these attributes MUST contain all information required to fulfill that individual subscription, even if that information is also sent in other attributes sent in fulfillment of other subscriptions at the same time. In other words, SW-PCs MUST NOT attempt to combine information when fulfilling multiple subscriptions simultaneously, even if this results in some redundancy in the attributes sent to the SW-PV.

3.6.5.5. Delayed Subscription Fulfillment

A SW-PC MAY delay the fulfillment of a subscription following a change event in the interest of waiting to see if additional change events are forthcoming and, if so, conveying the relevant records back to the SW-PV in a single SW Response attribute. This can help reduce network bandwidth consumption between the SW-PC and the SW-PV. For example, consider a situation where 10 changes occur a tenth of a second apart. If the SW-PC does not delay in assembling and sending SW Response attributes, the SW-PV will receive 10 separate SW Response attributes over a period of 1 second. However, if the SW-PC waits half a second after the initial event before assembling a SW Response, the SW-PV only receives two SW Response attributes over the same period of time.

Note that the ability to consolidate events for a single subscription over a given period of time does not contradict the rules in Section 3.6.5.4 prohibiting consolidation across multiple subscriptions. When delaying fulfillment of subscriptions, SW-PCs are still required to fulfill each individual subscription separately. Moreover, in the case that change events within the delay window cancel each other out (e.g., a record is deleted and then re-added), the SW-PC MUST still report each change event rather than just reporting the net effect of changes over the delay period. In other words, delayed fulfillment can decrease the number of attributes send by the SW-PC, but it does not reduce the total number of change events reported.

SW-PCs are not required to support delayed fulfillment of subscriptions. However, in the case that the SW-PC does support delayed subscription fulfillment, it MUST be possible to configure the SW-PC to disable delayed fulfillment. In other words, parties deploying SW-PCs need to be allowed to disable delayed subscription fulfillment in their SW-PCs. The manner in which such configuration occurs is left to the discretion of implementers, although implementers MUST protect the configuration procedure from unauthorized tampering. In other words, there needs to be some assurance that unauthorized individuals are not able to introduce long delays in subscription fulfillment.

3.7. Multiple Sources of Software Inventory Evidence Records

The records in an endpoint’s software inventory evidence collection might potentially come from multiple sources. For example, records might be derived from ISO SWID tags deposited on the file system and collected therefrom. Records might also be generated by tools such as software and package managers (e.g., RPM or YUM) or might be translated from software discovery reports.

A SW-PC is not required to identify every possible source of software information on its endpoint. Some SW-PCs might be explicitly tied only to one or a handful of software inventory sources. For all software inventory evidence sources that a particular SW-PC supports, it MUST completely support all requirements of this specification with regard to those sources. In other words, for supported sources, the SW-PC is required to be capable of providing a complete set of the provided software inventory evidence records; monitoring for changes in the records reported by those sources, correctly providing responses for both full and targeted requests for records from those sources, and delivering complete software inventory evidence records as appropriate. In all cases, the SW-PC MUST also be capable of deriving a Software Identifier from the resulting record and also assigning that record a unique Record Identifier. The SW-PC MUST NOT provide any inventory or event information from software inventory sources for which it cannot provide this full support. Note that the SW-PC SHOULD be able to provide a Software Locator for each software product reported by a given source, but it is recognized that this might not be possible in all circumstances and the inability to do so does not preclude use of the given source.

When providing a SW Response (either in direct response to a SW Request or in fulfillment of a subscription) the SW-PC MUST include the complete set of relevant data from all supported sources of software inventory evidence. In other words, a full inventory is required to contain all records from all supported sources, a targeted inventory is required to contain all relevant records from all sources, and event tracking is required to cover all events from all sources. With regard to events, a SW-PC’s assignment of EIDs MUST reflect the presence and order of all events on the endpoint (at least for supported sources) regardless of the source. This means that if source A experiences an event, and then source B experiences two events, and then source A experiences another two events, the SW-PC is required to capture five events with consecutive EID values reflecting the order in which the events occur.

Note that, if a SW-PC collects data from multiple sources, it is possible that some software products might be "double counted". This can happen if both sources of inventory evidence provide a record for a single installation of a software product. Moreover, each of these provided records might have different Software Identifier and Software Locator values due to the different ways a source might report its information. When a SW-PC reports information or records events from multiple inventory evidence sources, it MUST use the information those sources provide, rather than attempting to perform some form of reduction. In other words, if multiple sources report records corresponding to a single installation of a software product, all such records from each source are required to be part of the SW-

PC's processing even if this might lead to multiple reporting, and the SW-PC is not to ignore some records to avoid such multiple reporting. Similarly, in the case that multiple sources report an event, the SW-PC MUST create separate event records with separate EIDs for each of these, even if there is the chance that they represent multiple sources reporting the same action on the endpoint. Entities tracking software inventory information collected via SW-PCs and SW-PVs need to be aware that such double-reporting might occur. How (or if) such occurrences are detected and resolved is up to the implementers of those entities.

3.8. Error Handling

In the case where the SW-PC detects an error in a SW Request attribute that it receives it MUST respond with a PA-TNC Error attribute with an error code appropriate to the nature of the error. (See Section 4.2.8 of PA-TNC [RFC5792] for more details about PA-TNC Error attributes and error codes as well as Section 4.14 in this specification for error codes specific to SW Attributes.) In the case that an error is detected in a SW Request the SW-PC MUST NOT take any action requested by this SW Request, even if partial completion of the SW is possible. In other words, a SW Request that contains an error is completely ignored by the SW-PC (beyond sending a PA-TNC Error attribute, and possibly logging the error locally) rather than being partially executed.

In the case where the SW-PC receives a valid SW Request attribute but experiences an error during the process of responding to that attribute's instructions where that error prevents the SW-PC from properly or completely fulfilling that request, the SW-PC MUST send a PA-TNC Error attribute with an error code appropriate to the nature of the error. In the case where a PA-TNC Error attribute is sent, the SW-PC MUST NOT take any of the actions requested by the SW Request attribute which led to the detected error. This is the case even if some actions could have been completed successfully, and might even require the SW-PC to reverse some successful actions already taken before the error condition was detected. In other words, either all aspects of a SW Request complete fully and successfully (in which case the SW-PC sends a SW Response attribute), or no aspects of the SW Request occur (in which case the SW-PC sends a PA-TNC Error attribute). In the case that a SW-PC sends a PA-TNC Error attribute in response to a SW Request then the SW-PC MUST NOT also send any SW Response attribute in response to the same SW Request. For this reason, the sending of a SW Response attribute MUST be the last action taken by a SW-PC in response to a SW Request to avoid the possibility of a processing error occurring after that SW Response attribute is sent.

In the case that the SW-PC detects an error that prevents it from properly or completely fulfilling its obligations under an active subscription, the SW-PC MUST send a PA-TNC Error attribute of type SW_SUBSCRIPTION_FULFILLMENT_ERROR to the SW-PV that established this subscription. This type of PA-TNC Error attribute identifies the specific subscription that cannot be adequately honored due to the error condition as well as an error "sub-type". The error sub-type is used to indicate the type of error condition the SW-PC experienced that prevented it from honoring the given subscription. In the case that the error condition cannot be identified or does not align with any of the defined error codes, the SW_ERROR error code SHOULD be used in the sub-type. In the case that a SW_SUBSCRIPTION_FULFILLMENT_ERROR is sent, the associated subscription MUST be treated as cancelled by both the SW-PC and SW-PV.

The SW-PV MUST NOT send any PA-TNC Error attributes to SW-PCs. In the case that a SW-PV detects an error condition, it SHOULD log this error but does not inform any SW-PC’s of this event. Errors might include, but are not limited to, detection of malformed SW Response attributes sent from a given SW-PC, as well as detection of error conditions when the SW-PV processes SW Responses.

Both SW-PCs and SW-PVs SHOULD log errors so that administrators can trace the causes of errors. Log entries SHOULD include the type of the error, the time it was detected, and additional descriptive information to aid in understanding the nature and cause of the error.

4. Software Inventory Message and Attributes for PA-TNC Protocol

This section describes the format and semantics of the Software Inventory Message and Attributes for PA-TNC protocol. Software Inventory Message and Attributes for PA-TNC uses the standard PA-TNC message header format. See the PA-TNC specification [RFC5792] for information on this header format.

4.1. PA Subtype (AKA PA-TNC Component Type)

The NEA PB-TNC interface provides a general message-batching protocol capable of carrying one or more PA-TNC messages between the Posture Broker Client and Posture Broker Server. When PB-TNC is carrying a PA-TNC message, the PB-TNC message headers contain a 32 bit identifier called the PA Subtype. The PA Subtype field indicates the type of component associated with all of the PA-TNC attributes carried by the PB-TNC message. The core set of PA Subtypes is defined in the PA-TNC specification. This specification adds the following enumeration element to the IANA registry defined in section

7.2 of the PA-TNC specification [RFC5792] using the IETF Standard name space (SMI Private Enterprise Number 0x000000):

PEN	Integer	Name	Defining Specification
0	9	SW Attributes	Software Inventory Message and Attributes for PA-TNC

Table 2: PA Subtype

Each PA-TNC attribute described in this specification is intended to be sent between the SW-PC and SW-PV, so will be carried in a PB-TNC message indicating a PA Subtype of SW Attributes. Note that although the PA-TNC Error attribute is defined in the PA-TNC specification, when it is used in a SW Attribute exchange, it uses the SW Attributes Component Definition Value, as described in Section 4.2.8 of the PA-TNC specification [RFC5792]. PB-TNC messages MUST always include the SW Attributes Subtype defined in this section when carrying SW Attributes over PA-TNC.

For more information on PB-TNC and PA-TNC messages and message headers, see the PB-TNC [RFC5793] and PA-TNC [RFC5792] specifications, respectively.

4.2. SW Attribute Overview

The attributes defined in this specification appear below with a short summary of their purposes. Each attribute is described in greater detail in subsequent sections.

- SW Request - This attribute is used to request a software inventory or software event list from an endpoint. This attribute might also establish a subscription on the recipient SW-PC. A SW-PC MUST NOT send this attribute.

- Software Identifier Inventory - This attribute is used to convey an inventory without the inclusion of software inventory evidence records. When a SW-PC receives a SW Request attribute requesting an inventory without software inventory evidence records, the SW-PC MUST send a Software Identifier Inventory attribute (or a PA-TNC Error) in response. This attribute also MAY be sent by the SW-PC in fulfillment of an active subscription. A SW-PV MUST NOT send this attribute.

- Software Identifier Events - This attribute is used to convey a list of events concerning changes to an endpoint’s Software

Inventory Evidence Collection. Reported events do not include software inventory evidence records. When a SW-PC receives a SW Request attribute requesting an event collection without software inventory evidence records, the SW-PC MUST send a Software Identifier Events attribute (or a PA-TNC Error) in response. This attribute also MAY be sent by the SW-PC in fulfillment of an active subscription. A SW-PV MUST NOT send this attribute.

- Software Inventory - This attribute is used to convey an inventory expressed including software inventory evidence records. When a SW-PC receives a SW Request attribute requesting an inventory including software inventory evidence records, the SW-PC MUST send a Software Inventory attribute (or a PA-TNC Error) in response. This attribute also MAY be sent by the SW-PC in fulfillment of an active subscription. A SW-PV MUST NOT send this attribute.

- Software Events - This attribute is used to convey a list of events concerning changes to an endpoint’s inventory evidence collection. Reported events include software inventory evidence records. When a SW-PC receives a SW Request attribute requesting an event collection including software inventory evidence records, the SW-PC MUST send a Software Events attribute (or a PA-TNC Error) in response. This attribute also MAY be sent by the SW-PC in fulfillment of an active subscription. A SW-PV MUST NOT send this attribute.

- Subscription Status Request - This attribute is used to request a SW-PC send a summary of all the active subscriptions it has where the requesting party is the subscriber. The SW-PC MUST respond with a Subscription Status Response (or a PA-TNC Error). A SW-PC MUST NOT send this attribute.

- Subscription Status Response - This attribute is used to convey information about the active subscriptions that a SW-PC has for a given subscriber. A SW-PV MUST NOT send this attribute.

- PA-TNC Error - This is the standard PA-TNC Error attribute as defined in PA-TNC [RFC5792] and is used to indicate that an error was encountered during a SW Attribute exchange. It MUST be sent by a SW-PC in response to a SW Request in the case where the SW-PC encounters a fatal error (i.e., an error that prevents further processing of an exchange) relating to the attribute exchange. A SW-PV MUST NOT send this attribute. The SW-PC MUST then ignore the erroneous attribute after a PA-TNC Error attribute is sent (i.e., do not attempt to act on an attribute that generated a PA-TNC Error beyond sending the PA-TNC Error). In the case where the SW-PV experiences a fatal error, it MUST ignore the erroneous attribute without sending a PA-TNC Error attribute. It MAY take

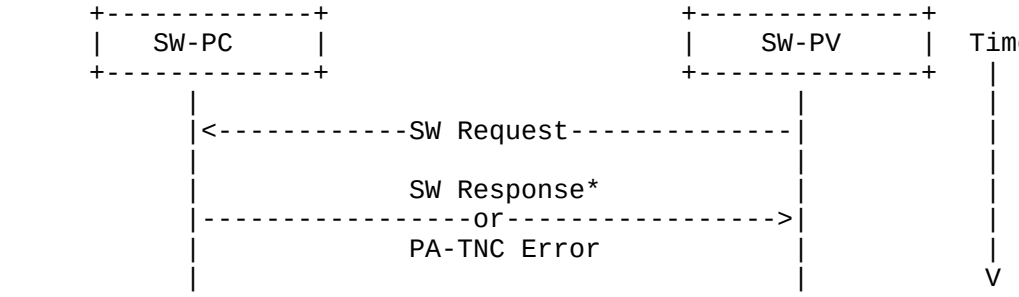
other actions in response to the error, such as logging the cause of the error, or even taking actions to isolate the endpoint

Because one of the Software Identifier Inventory, Software Identifier Events, Software Inventory, or Software Events attributes is expected to be sent to a SW-PV in direct response to a SW Request attribute or in fulfillment of an active subscription, those four attribute types are referred to collectively in this document as "SW Response" attributes.

All SW-PVs MUST be capable of sending SW Requests and be capable of receiving and processing all SW Response attributes as well as PA-TNC Error attributes. All SW-PCs MUST be capable of receiving and processing SW Requests and be capable of sending all types of SW Response attributes as well as PA-TNC Error attributes. In other words, both SW-PVs and SW-PCs are required to support their role in exchanges using any of the attribute types defined in this section. SW-PVs MUST ignore any SW Request attributes that they receive. SW-PCs MUST ignore any SW Response attributes or PA-TNC Error attributes that they receive.

4.3. SW Attribute Exchanges

A SW Attribute Exchange is used to provide the SW-PV with a software inventory or event collection from the queried endpoint.



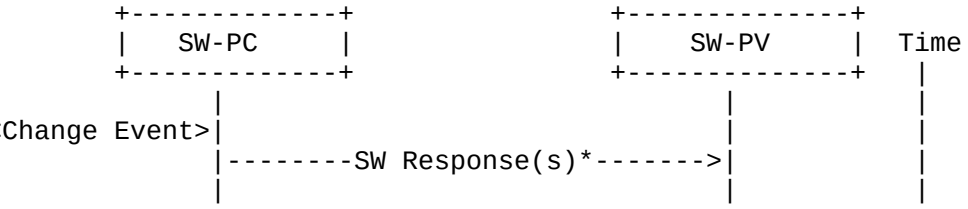
*SW Response is one of the following: Software Identifier Inventory, Software Identifier Events, Software Inventory, or Software Events.

Figure 4: SW Attribute Exchange (Direct Response to SW Request)

In this exchange, the SW-PV indicates to the SW-PC, via a SW Request, the nature of the information it wishes to receive (inventory vs. events, full or targeted) and how it wishes the returned inventory to

be expressed (with or without software inventory evidence records). The SW-PC responds with the requested information using the appropriate attribute type. A single SW Request MUST only lead to a single SW Response or PA-TNC Error that is in direct response to that request.

In addition, if there is an active subscription on the endpoint, the SW-PC sends a SW Response to the SW-PV following a change event on the endpoint in fulfillment of that subscription. Such an exchange is shown in Figure 5.



*SW Response is one of the following: Software Identifier Inventory, Software Identifier Events, Software Inventory, or Software Events.

Figure 5: SW Attribute Exchange (In Fulfillment of an Active Subscription)

Note that, unlike direct responses to a SW Request, a single change event can precipitate multiple SW Responses for a single subscription, but only if all but the last of those SW Responses convey partial lists of event records, and the last of those SW Responses conveys a complete list of event records. (That is, the initial responses are partial lists and the last response is the remainder of the relevant event records, completing the delivery of all relevant events at the time of the change event.) A single Change Event MUST NOT otherwise be followed by multiple SW Response or PA-TNC Error attributes in any combination.

All SW-PVs and SW-PCs MUST support both types of exchanges. In particular, SW-PCs MUST be capable of pushing a SW Response to a SW-PV immediately upon detection of a change to the endpoint's Software Inventory Evidence Collection in fulfillment of established SW-PV subscriptions, as described in Section 3.6.

4.4. Software Inventory Message and Attributes for PA-TNC Attribute Enumeration

PA-TNC attribute types are identified in the PA-TNC Attribute Header via the Attribute Type Vendor ID and Attribute Type fields. Table 3 identifies the appropriate values for these fields for each attribute type used within the Software Inventory Message and Attributes for PA-TNC protocol.

Attribute Name	PEN	Integer	Description
SW Request	0x000000	0x00000011	Request from a SW-PV to a SW-PC for the SW-PC to provide a software inventory or event list
Software Identifier Inventory	0x000000	0x00000012	An inventory sent without softare inventory evidence records sent from a SW-PC.
Software Identifier Events	0x000000	0x00000013	A collection of events impacting the endpoint’s Software Inventory Evidence Collection, where events do not include software inventory evidence records.
Software Inventory	0x000000	0x00000014	An inventory including software inventory evidence records sent from a SW-PC.
Software Events	0x000000	0x00000015	A collection of events impacting the endpoint’s Software Inventory Evidence Collection, where events include software inventory evidence records.
Subscription Status Request	0x000000	0x00000016	A request for a list of a SW-PV’s active subscription.
Subscription	0x000000	0x00000017	A list of a SW-PV’s active

Status Response			subscriptions.
Reserved	0x000000	0x00000018 - 0x0000001F	These attribute types are reserved for future use in revisions to Software Inventory Message and Attributes for PA-TNC.
PA-TNC Error	0x000000	0x00000008	An error attribute as defined in the PA-TNC specification [RFC5792].

Table 3: SW Attribute Enumeration

4.5. Normalization of Text Encoding

In order to ensure consistency of transmitted attributes some fields require normalization of their format. When this is necessary, this is indicated in the field's description. In such cases, the field contents MUST be normalized to Network Unicode format as defined in RFC 5198 [RFC5198]. Network Unicode format defines a refinement of UTF-8 that ensures a normalized expression of characters. SW-PCs and SW-PVs MUST NOT perform conversion and normalization on any field values except those specifically identified as requiring normalization in the following sections. Note, however, that some data models require additional normalization before source information is added to an Endpoint's Inventory Evidence Collection as a record. The references from the Software Data Model IANA table (see Section 9.4) will note where this is necessary.

4.6. Request IDs

All SW Request attributes MUST include a Request ID value. The Request ID field provides a value that identifies a given request relative to other requests between a SW-PV and the receiving SW-PC. Specifically, the SW-PV assigns each SW Request attribute a Request ID value that is intended to be unique within the lifetime of a given network connection ID as assigned by the SW-PV's Posture Broker Server. In the case where all possible Request ID values have been exhausted within the lifetime of a single network connection ID, the sender MAY reuse previously used Request IDs within the same network connection that are not being used as Subscription IDs. (See below in this section for an explanation of Subscription ID assignment.) In this case of Request ID reuse, Request IDs SHOULD be reused in the order of their original use. In other words, a SW-PC SHOULD NOT use a given Request ID value more than once within a persistent

connection between a given Posture Broker Client-Posture Broker Server pair, but, in the case where reuse is necessary due to exhaustion of possible ID values, the SW-PC SHOULD structure the reuse to maximize the time between original and subsequent use. The Request ID value is included in a SW Response attribute directly responding to this SW Request to indicate which SW Request was received and caused the response. Request IDs can be randomly generated or sequential, as long as values are not repeated per the rules in this paragraph. SW-PCs are not required to check for duplicate Request IDs.

In the case that a SW Request requests the establishment of a subscription and the receiving SW-PC agrees to that subscription, the Request ID of that SW Request (i.e., the establishing request of the subscription) becomes that subscription's Subscription ID. All attributes sent in fulfillment of this subscription include a flag indicating that the attribute fulfills a subscription and the subscription's Subscription ID. A SW-PV MUST NOT reuse a Request ID value in communicating to a given SW-PC while that Request ID is also serving as a Subscription ID for an active subscription with that SW-PC. In the case where a SW-PC receives a SW Request from a given SW-PV where that Request ID is also the Subscription ID of an active subscription with that SW-PV, the SW-PC MUST respond with a PA-TNC Error attribute with an error code of SW_SUBSCRIPTION_ID_REUSE_ERROR. Note that this error does not cancel the indicated subscription.

Subscription Status Requests and Subscription Status Responses do not include Request IDs.

4.7. SW Request

A SW-PV sends this attribute to a SW-PC to request that the SW-PC send software inventory information to the SW-PV. A SW-PC MUST NOT send this attribute.

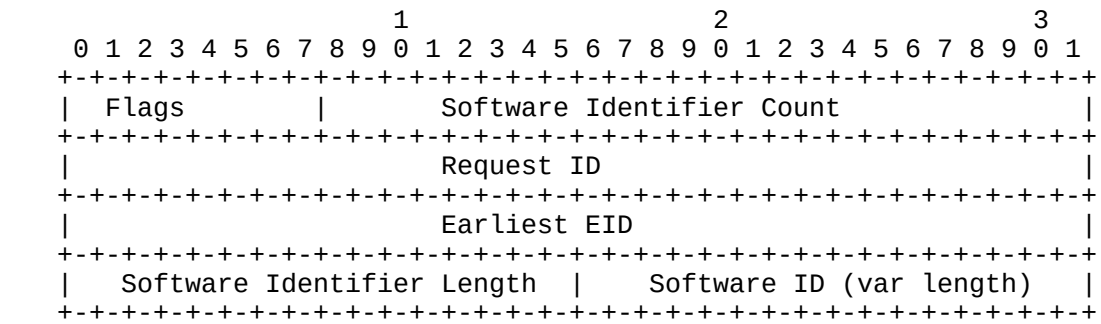


Figure 6: SW Request Attribute

Field	Description
Flags: Bit 0 - Clear Subscriptions	If set (1), the SW-PC MUST delete all subscriptions established by the requesting SW-PV (barring any errors).
Flags: Bit 1 - Subscribe	If set (1), in addition to responding to the request as described, the SW-PC MUST establish a subscription with parameters matching those in the request attribute (barring any errors).
Flags: Bit 2 - Result Type	If unset (0), the SW-PC's response MUST include software inventory evidence records and thus the response MUST be a Software Inventory, a Software Events, or a PA-TNC Error attribute. If set (1), the response MUST NOT include software inventory evidence records and thus the response MUST be a Software Identifier Inventory, a Software Identifier Events, or a PA-TNC Error attribute.
Flags: Bit 3-7 - Reserved	Reserved for future use. This field MUST be set to zero on transmission and ignored upon reception.
Software Identifier Count	A 3-byte unsigned integer indicating the number of Software Identifiers that follow. If this value is non-zero, this is a targeted request, as described in Section 3.3. The Software Identifier Length and Software ID fields are repeated, in order, the number of times indicated in this field. In the case where Software

		Identifiers are present, the SW-PC MUST only report software that corresponds to the identifiers the SW-PV provided in this attribute (or with a PA-TNC Error attribute). This field value MAY be 0, in which case there are no instances of the Software Identifier Length and Software ID fields. In this case, the SW-PV is indicating an interest in all software inventory evidence records on the endpoint (i.e., this is not a targeted request).
	Request ID	A value that uniquely identifies this SW Request from a particular SW-PV.
	Earliest EID	In the case where the SW-PV is requesting software events, this field contains the EID value of the earliest event the SW-PV wishes to have reported. (Note - the report will be inclusive of the event with this EID value.) In the case where the SW-PV is requesting an inventory, then this field MUST be 0. (0x00000000) In the case where this field is non-zero, the SW-PV is requesting events and the SW-PC MUST respond using a Software Events, Software Identifier Events, or a PA-TNC Error attribute. In the case where this field is zero, the SW-PV is requesting an inventory and the SW-PC MUST respond using a Software Inventory, a Software Identifier Inventory, or a PA-TNC Error attribute.
	Software Identifier Length	A 2-byte unsigned integer indicating the length in bytes of the Software ID field.
	Software ID	A string containing the Software Identifier value from a software inventory evidence record. This field value MUST be normalized to Network Unicode format, as described in Section 4.5. This string MUST NOT be NULL terminated.

Table 4: SW Request Attribute Fields

The SW-PV sends the SW Request attribute to a SW-PC to request the indicated information. Note that between the Result Type flag and the Earliest EID field, the SW-PC is constrained to a single possible

SW Response attribute type (or a PA-TNC Error attribute) in its response to the request.

The Subscribe and Clear Subscription flags are used to manage subscriptions for the requesting SW-PV on the receiving SW-PC. Specifically, an attribute with the Subscribe flag set seeks to establish a new subscription by the requesting SW-PV to the given SW-PC, while an attribute with the Clear Subscription flag seeks to delete all existing subscriptions by the requesting SW-PV on the given SW-PC. Note that, in the latter case, only the subscriptions associated with the Connection ID and the Posture Validator ID of the requester are deleted as described in Section 3.6.3. A newly established subscription has the parameters outlined in the Request attribute. Specifically, the Result Type flag indicates the type of result to send in fulfillment of the subscription, the value of the Earliest EID field indicates whether the fulfillment attributes list inventories or events, and the fields describing Software Identifiers (if present) indicate if and how a subscription is targeted. In the case that the SW-PC is unable or unwilling to comply with the SW-PV's request to establish or clear subscriptions, the SW-PC MUST respond with a PA-TNC Error attribute with the SW_SUBSCRIPTION_DENIED_ERROR error code. (Note that if the SW-PV requests that subscriptions be cleared but has no existing subscriptions, this is not an error.)

An attribute requesting the establishment of a subscription is effectively doing double-duty, as it is a request for an immediate response from the SW-PC in addition to setting up the subscription. Assuming the SW-PC is willing to comply with the subscription it MUST send an appropriate response attribute to a request with the Subscribe flag set containing all requested information. The same is true of the Clear Subscription flag - assuming there is no error the SW-PC MUST generate a response attribute without regard to the presence of this flag in addition to clearing its subscription list.

Both the Subscribe and Clear Subscription flags MAY be set in a single SW Request attribute. In the case where this request is successful, the end result MUST be equivalent to the SW-PC clearing its subscription list for the given SW-PV first and then creating a new subscription in accordance with the request parameters. (In other words, do not first create the new subscription and then clear all the subscriptions including the one that was just created.) In the case that the requested actions are successfully completed, the SW-PC MUST respond with a SW Response attribute. (The specific type of SW Response attribute depends on the Result Type and Earliest EID fields, as described above.) In the case where there is a failure that prevents some part this request from completing, the SW-PC MUST NOT add a new subscription, MUST NOT clear the old subscriptions, and the SW-PC MUST respond with a PA-TNC Error attribute. In other

words, the SW-PC MUST NOT partially succeed at implementing such a request; either all actions succeed, or none succeed.

The Earliest EID field is used to indicate whether the SW-PV is requesting an inventory or event list from the SW-PC. A value of 0 (0x00000000) represents a request for inventory information. Otherwise, the SW-PV is requesting event information. For Earliest EID values other than 0, the SW-PC's response MUST respond with event records, as described in Section 3.5. Note that the request does not identify a particular EID Epoch, since responses can only include events in the SW-PC's current EID Epoch.

The Software Identifier Count indicates the number of Software Identifiers in the attribute. This number might be any value between 0 and 16,777,216, inclusive. A single Software Identifier is represented by the following fields: Software Identifier Length and Software ID. These fields are repeated a number of times equal to the Software Identifier Count. Note that this could be 0 times. The Software Identifier Length field indicates the number of bytes allocated to the Software ID field. The Software Identifier field contains a Software Identifier as describe in Section 3.2.1. The presence of one or more Software Identifiers is used by the SW-PV to indicate a targeted request, which seeks only inventories of or events affecting software corresponding to the given identifiers. The SW-PC MUST only report software that matched the Software Identifiers provided in the SW-PVs SW Request attribute.

4.8. Software Identifier Inventory

A SW-PC sends this attribute to a SW-PV to convey the inventory of the endpoint's Software Inventory Evidence Collection without the inclusion of software inventory evidence records. This list might represent a complete inventory or a targeted list of records, depending on the parameters in the SW-PV's request. A SW-PV MUST NOT send this attribute. The SW-PC either sends this attribute in fulfillment of an existing subscription where the establishing request has a Result Type of 1 and the Earliest EID is zero, or in direct response to a SW Request attribute where the Result Type is 1 and the Earliest EID is zero.

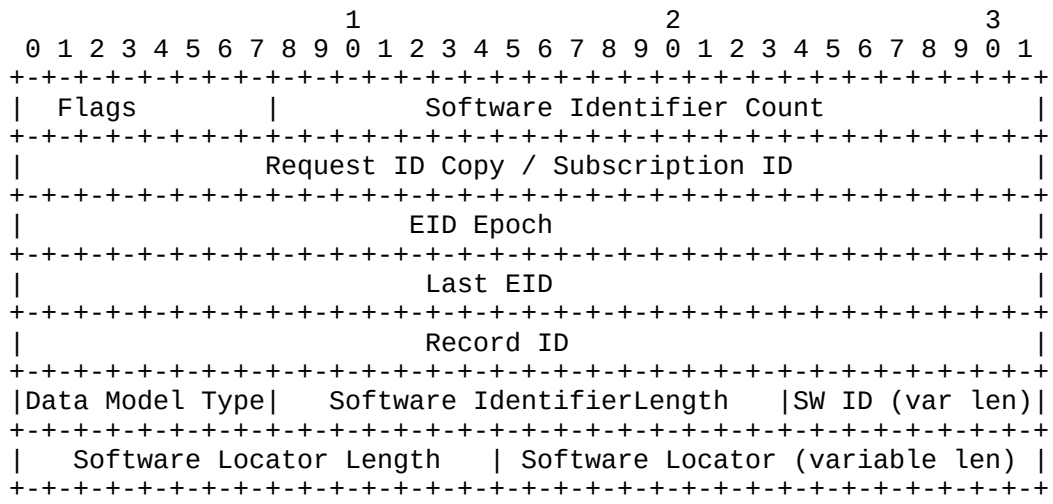


Figure 7: Software Identifier Inventory Attribute

Field	Description
Flags: Bit 0 - Subscription Fulfillment	In the case that this attribute is sent in fulfillment of a subscription this bit MUST be set (1). In the case that this attribute is a direct response to a SW Request, this bit MUST be unset (0).
Flags: Bit 1-7 - Reserved	Reserved for future use. This field MUST be set to zero on transmission and ignored upon reception.
Software Identifier Count	The number of Software Identifiers that follow. This field is an unsigned integer. The Record ID, Data Model Type, Software Identifier Length, SW ID, Software Locator Length, and Software Locator fields are repeated, in order, the number of times indicated in this field. This field value MAY be 0, in which case there are no instances of these fields.
Request ID Copy / Subscription ID	In the case where this attribute is in direct response to a SW Request attribute from a SW-PV, this field MUST contain an exact copy of the Request ID field from that SW Request. In the

	case where this attribute is sent in fulfillment of an active subscription, this field MUST contain the Subscription ID of the subscription being fulfilled by this attribute.
EID Epoch	The EID Epoch of the Last EID value. This field is an unsigned 4-byte integer.
Last EID	The EID of the last event recorded by the SW-PC, or 0 if the SW-PC has no recorded events. This field is an unsigned 4-byte integer.
Record ID	A 4-byte, unsigned integer containing the Record Identifier value from a software inventory evidence record.
Data Model Type	A 1-byte unsigned integer containing an identifier number from the Software Data Model IANA table that identifies the data model of the reported record.
Software Identifier Length	A 2-byte unsigned integer indicating the length in bytes of the SW ID field.
SW ID	A string containing the Software Identifier value from a software inventory evidence record. This field value MUST be normalized to Network Unicode format, as described in Section 4.5. This string MUST NOT be NULL terminated.
Software Locator Length	A 2-byte unsigned integer indicating the length in bytes of the Software Locator field.
Software Locator	A string containing the Software Locator value. This is expressed as a URI. This field value MUST be normalized to Network Unicode format as described in Section 3.2.1.2. This string MUST NOT be NULL terminated.

Table 5: Software Identifier Inventory Attribute Fields

In the case that this attribute is sent in fulfillment of a subscription, the Subscription Fulfillment bit MUST be set (1). In the case that this attribute is sent in direct response to a SW Request, the Subscription Fulfillment bit MUST be unset (0). Note

that the SW Response attribute sent in direct response to a SW Request that establishes a subscription (i.e., a subscription's establishing request) MUST be treated as a direct response to that SW Request (and thus the Subscription Fulfillment bit is unset). SW Response attributes are only treated as being in fulfillment of a subscription (i.e., Subscription Fulfillment bit set) if they are sent following a change event, as shown in Figure 3.

The Software Identifier Count field indicates the number of Software Identifiers present in this inventory. Each Software Identifier is represented by the following set of fields: Record ID, Data Model Type, Software Identifier Length, SW ID, Software Locator Length, and Software Locator. These fields will appear once for each reported record.

When responding directly to a SW Request attribute, the Request ID Copy / Subscription ID field MUST contain an exact copy of the Request ID field from that SW Request. When this attribute is sent in fulfillment of an existing subscription on this Posture Collector, then this field MUST contain the Subscription ID of the fulfilled subscription.

The EID Epoch field indicates the EID Epoch of the Last EID value. The Last EID field MUST contain the EID of the last recorded change event (see Section 3.5 for more about EIDs and recorded events) at the time this inventory was collected. In the case where there are no recorded change events at the time that this inventory was collected, the Last EID field MUST contain 0. These fields can be interpreted to indicate that the provided inventory reflects the state of the endpoint after all changes up to and including this last event have been accounted for.

4.9. Software Identifier Events

A SW-PC sends this attribute to a SW-PV to convey events where the affected records are reported without software inventory evidence records. A SW-PV MUST NOT send this attribute. The SW-PC either sends this attribute in fulfillment of an existing subscription where the establishing request has a Result Type is 1 and the Earliest EID is non-zero, or in direct response to a SW Request attribute where the Result Type is 1 and the Earliest EID is non-zero.

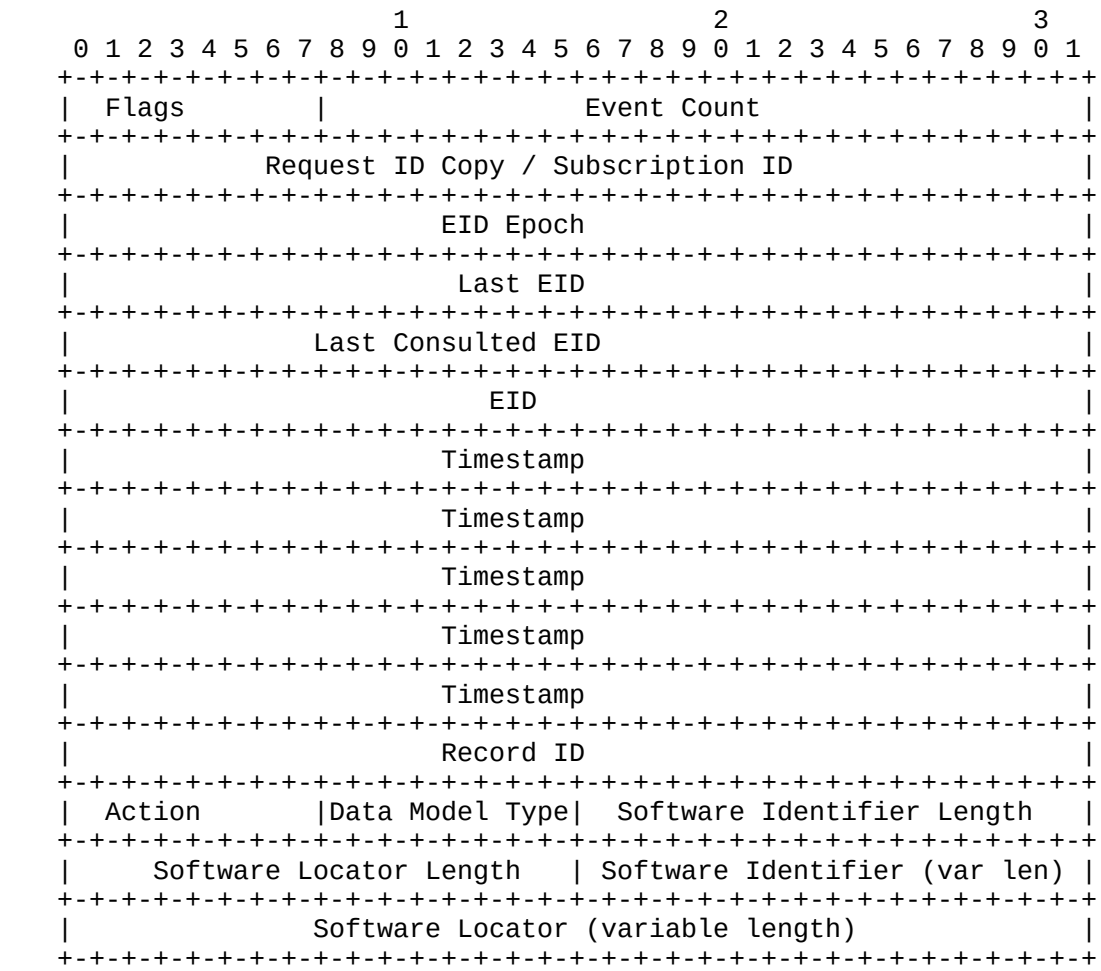


Figure 8: Software Identifier Events Attribute

Field	Description
Flags: Bit 0 - Subscription Fulfillment	In the case that this attribute is sent in fulfillment of a subscription this bit MUST be set (1). In the case that this attribute is a direct response to a SW Request, this bit MUST be unset (0).
Flags: Bit 1-7 -	Reserved for future use. This field MUST be set to zero on transmission and ignored upon reception.

Reserved	
Event Count	The number of events that are reported in this attribute. This field is a 3-byte unsigned integer. The EID, Timestamp, Record ID, Action, Data Model Type, Software Identifier Length, Software Locator Length, Software Identifier, and Software Locator fields are repeated, in order, the number of times indicated in this field. This field value MAY be 0, in which case there are no instances of these fields.
Request ID Copy / Subscription ID	In the case where this attribute is in direct response to a SW Request attribute from a SW-PV, this field MUST contain an exact copy of the Request ID field from that SW Request. In the case where this attribute is sent in fulfillment of an active subscription, this field MUST contain the Subscription ID of the subscription being fulfilled by this attribute.
EID Epoch	The EID Epoch of the Last EID value. This field is an unsigned 4-byte integer.
Last EID	The EID of the last event recorded by the SW-PC, or 0 if the SW-PC has no recorded events. This field contains the EID of the SW-PC's last recorded change event (which might or might not be included as an event record in this attribute).
Last Consulted EID	The EID of the last event record that was consulted when generating the event record list included in this attribute. This is different from the Last EID field value if and only if this attribute is conveying a partial list of event records. See Section 3.5.5 for more on partial list of event records.
EID	The EID of the event in this event record.
Timestamp	The timestamp associated with the event in this event record. This timestamps is the SW-PC's best understanding of when the given event occurred. Note that this timestamp might be an estimate. The Timestamp date and time MUST be represented as an RFC 3339 [5] compliant ASCII string in Coordinated Universal Time (UTC) time with the additional restrictions that the 'T' delimiter and

		the 'Z' suffix MUST be capitalized and fractional seconds (time-secfrac) MUST NOT be included. This field conforms to the date-time ABNF production from section 5.6 of RFC 3339 [RFC3339] with the above restrictions. Leap seconds are permitted and SW-PVs MUST support them. The Timestamp string MUST NOT be NULL terminated or padded in any way. The length of this field is always 20 octets.
Record ID		A 4-byte, unsigned integer containing the Record Identifier value from a software inventory evidence record.
Action		The type of event that is recorded in this event record. Possible values are: 1 = CREATION - the addition of a record to the endpoint's Software Inventory Evidence Collection; 2 = DELETION - the removal of a record from the endpoint's Software Inventory Evidence Collection; 3 = ALTERATION - There was an alteration to a record within the endpoint's Software Inventory Evidence Collection. All other values are reserved for future use and MUST NOT be used when sending attributes. In the case where a SW-PV receives an event record that uses an action value other than the ones defined here, it MUST ignore that event record but SHOULD process other event records in this attribute as normal.
Data Model Type		A 1-byte unsigned integer containing an identifier number from the Software Data Model IANA table that identifies the data model of the reported record.
Software Identifier Length		A 2-byte unsigned integer indicating the length in bytes of the Software Identifier field.
Software Locator Length		A 2-byte unsigned integer indicating the length in bytes of the Software Locator field.
Software Identifier		A string containing the Software Identifier value from a software inventory evidence record. This field value MUST be normalized to Network Unicode format, as described in Section 4.5. This string MUST NOT be NULL terminated.

Software Locator	A string containing the Software Locator value. This is expressed as a URI. This field value MUST be normalized to Network Unicode format as described in Section 3.2.1.2. This string MUST NOT be NULL terminated.

Table 6: Software Identifier Events Attribute Fields

The first few fields in the Software Identifier Events attribute mirror those in the Software Identifier Inventory attribute. The primary difference is that, instead of conveying an inventory, the attribute conveys zero or more event records, consisting of the EID, timestamp, Record ID, action type, data model type, Software Identifier, and Software Locator of the affected software inventory evidence record.

With regard to the Timestamp field, it is important to note that clock skew between the SW-PC and SW-PV as well as between different SW-PCs within an enterprise might make correlation of timestamp values difficult. This specification does not attempt to resolve clock skew issues, although other mechanisms outside of this specification do exist to reduce the impact of clock skew and make the timestamp more useful for such correlation. Instead, Software Inventory Message and Attributes for PA-TNC uses Timestamp value primarily as a means to indicate the amount of time between two events on a single endpoint. For example, by taking the difference of the times for when a record was removed and then subsequently re-added, one can get an indication as to how long the system was without the given record (and, thus without the associated software). Since this will involve comparison of timestamp values all originating on the same system, clock skew between the SW-PC and SW-PV is not an issue. However, if the SW-PC’s clock was adjusted between two recorded events, it is possible for such a calculation to lead to incorrect understandings of the temporal distance between events. Users of this field need to be aware of the possibility for such occurrences. In the case where the Timestamp values of two events appear to contradict the EID ordering of those events (i.e., the later EID has an earlier timestamp) the recipient MUST treat the EID ordering as correct.

All events recorded in a Software Identifier Events Attribute are required to be part of the same EID Epoch. Specifically, all reported events MUST have an EID from the same EID Epoch as each other, and which is the same as the EID Epoch of the Last EID and Last Consulted EID values. The SW-PC MUST NOT report events with EIDs from different EID Epochs.

The Last Consulted EID field contains the EID of the last event record considered for inclusion in this attribute. If this attribute contains a partial event set (as described in Section 3.5.5) this field value will be less than the Last EID value; if this attribute contains a complete event set, the Last EID and Last Consulted EID values are identical.

If multiple events are sent in a Software Identifier Events attribute, the order in which they appear within the attribute is not significant. The EIDs associated with them are used for ordering the indicated events appropriately. Also note that a single software record might be reported multiple times in an attribute, such as if multiple events involving the associated record were being reported.

4.10. Software Inventory

A SW-PC sends this attribute to a SW-PV to convey a list of inventory records. A SW-PV MUST NOT send this attribute. The SW-PC either sends this attribute in fulfillment of an existing subscription where the establishing request had a Result Type of 0 and the Earliest EID is zero, or in direct response to a SW Request attribute where the Result Type is 0 and the Earliest EID is zero.

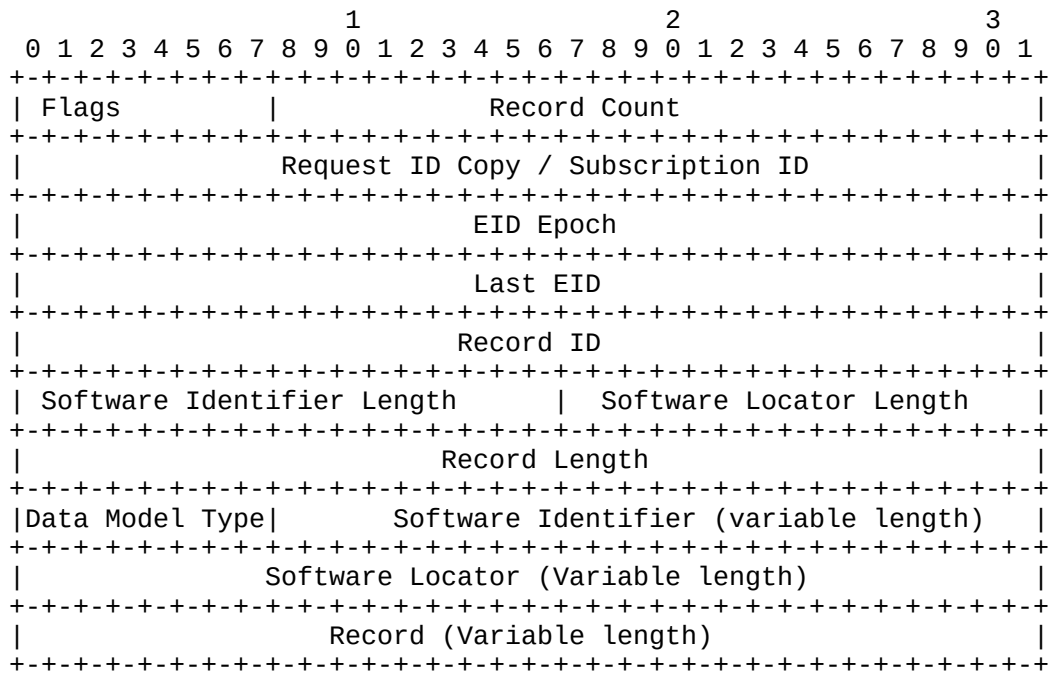


Figure 9: Software Inventory Attribute

Field	Description
Flags: Bit 0 - Subscription Fulfillment	In the case that this attribute is sent in fulfillment of a subscription this bit MUST be set (1). In the case that this attribute is a direct response to a SW Request, this bit MUST be unset (0).
Flags: Bit 1-7 - Reserved	Reserved for future use. This field MUST be set to zero on transmission and ignored upon reception.
Record Count	The number of records that follow. This field is a 3-byte unsigned integer. The Record ID, Software Identifier Length, Software Locator Length, Record Length, Data Model Type, Software Identifier, Software Locator, and Record fields are repeated, in order, the number of times indicated in this field. This field value MAY be 0 in which case there are no instances of these fields.

Request ID Copy / Subscription ID	In the case where this attribute is in direct response to a SW Request attribute from a SW-PV, this field MUST contain an exact copy of the Request ID field from that SW Request. In the case where this attribute is sent in fulfillment of an active subscription, this field MUST contain the Subscription ID of the subscription being fulfilled by this attribute.
EID Epoch	The EID Epoch of the Last EID value. This field is an unsigned 4-byte integer.
Last EID	The EID of the last event recorded by the SW-PC, or 0 if the SW-PC has no recorded events. This field is an unsigned 4-byte integer.
Record ID	A 4-byte, unsigned integer containing the Record Identifier value from a software inventory evidence record.
Software Identifier Length	A 2-byte unsigned integer indicating the length in bytes of the Software Identifier field.
Software Locator Length	A 2-byte unsigned integer indicating the length in bytes of the Software Locator field.
Record Len	This is a 4-byte unsigned integer indicating the length of the Record field in bytes.
Data Model Type	A 1-byte unsigned integer containing an identifier number from the Software Data Model IANA table that identifies the data model of the reported record.
Software Identifier	A string containing the Software Identifier value from a software inventory evidence record. This field value MUST be normalized to Network Unicode format, as described in Section 4.5. This string MUST NOT be NULL terminated.
Software Locator	A string containing the Software Locator value. This is expressed as a URI. This field value MUST be normalized to Network Unicode format as described in Section 3.2.1.2. This string MUST NOT be NULL terminated.

Record	A software inventory evidence record as a string. The record MUST be converted and normalized to Network Unicode format, as described in Section 4.5. This string MUST NOT be NULL terminated.

Table 7: Software Inventory Attribute Fields

The Software Inventory attribute contains some number of software inventory evidence records along with the core response attribute fields. Given that the size of records can vary considerably, the length of this attribute is highly variable and, if transmitting a complete inventory, can be extremely large. Enterprises might wish to constrain the use of Software Inventory attributes to targeted requests to avoid over-burdening the network unnecessarily.

When copying a software inventory evidence record into the Record field, the record MUST be converted and normalized to use Network Unicode format prior to its inclusion in the record field.

4.11. Software Events

A SW-PC sends this attribute to a SW-PV to convey a list of events that include software inventory evidence records. A SW-PV MUST NOT send this attribute. The SW-PC either sends this attribute in fulfillment of an existing subscription where the establishing request has a Result Type of 0 and the Earliest EID is non-zero, or in direct response to a SW Request attribute where the Result Type is 0 and the Earliest EID is non-zero.

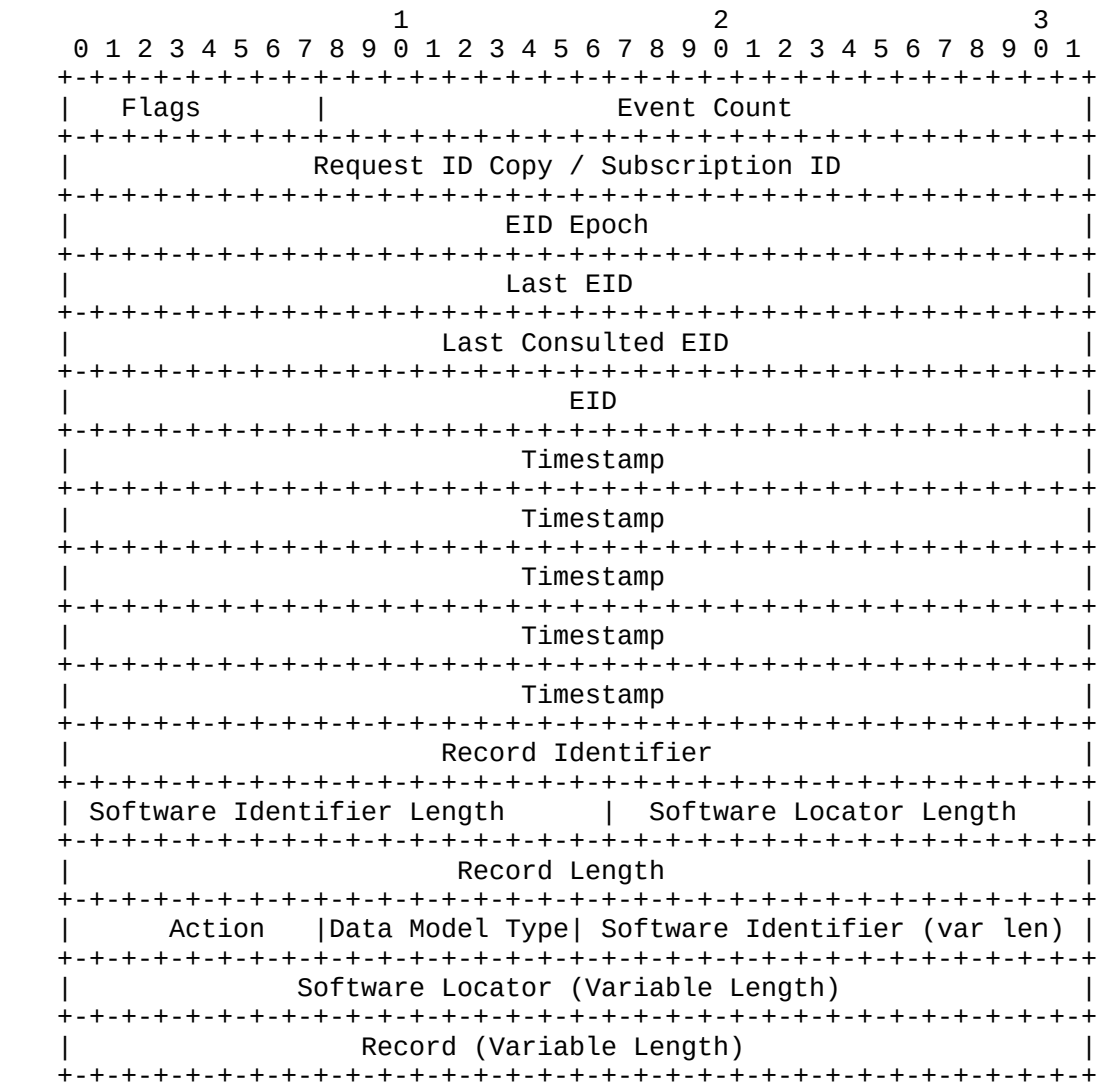


Figure 10: Software Events Attribute

Field	Description
Flags: Bit 0 - Subscription Fulfillment	In the case that this attribute is sent in fulfillment of a subscription this bit MUST be set (1). In the case that this attribute is a direct response to a SW Request, this bit MUST be unset

	(0).
Flags: Bit 1-7 - Reserved	Reserved for future use. This field MUST be set to zero on transmission and ignored upon reception.
Event Count	The number of events being reported in this attribute. This field is a 3-byte unsigned integer. The EID, Timestamp, Record Identifier, Software Identifier Length, Software Locator Length, Record Length, Action, Data Model Type, Software Identifier, Software Locator, and Record fields are repeated, in order, the number of times indicated in this field. This field value MAY be 0, in which case there are no instances of these fields.
Request ID Copy / Subscription ID	In the case where this attribute is in direct response to a SW Request attribute from a SW-PV, this field MUST contain an exact copy of the Request ID field from that SW Request. In the case where this attribute is sent in fulfillment of an active subscription, this field MUST contain the Subscription ID of the subscription being fulfilled by this attribute.
EID Epoch	The EID Epoch of the Last EID value. This field is an unsigned 4-byte integer.
Last EID	The EID of the last event recorded by the SW-PC, or 0 if the SW-PC has no recorded events. This field contains the EID of the SW-PC's last recorded change event (which might or might not be included as an event record in this attribute).
Last Consulted EID	The EID of the last event record that was consulted when generating the event record list included in this attribute. This is different from the Last EID field value if and only if this attribute is conveying a partial list of event records. See Section 3.5.5 for more on partial list of event records.
EID	The EID of the event in this event record.
Timestamp	The timestamp associated with the event in this event record. This timestamp is the SW-PC's best understanding of when the given event occurred.

	Note that this timestamp might be an estimate. The Timestamp date and time MUST be represented as an RFC 3339 [5] compliant ASCII string in Coordinated Universal Time (UTC) time with the additional restrictions that the 'T' delimiter and the 'Z' suffix MUST be capitalized and fractional seconds (time-secfrac) MUST NOT be included. This field conforms to the date-time ABNF production from section 5.6 of RFC 3339 [RFC3339] with the above restrictions. Leap seconds are permitted and SW-PVs MUST support them. The Timestamp string MUST NOT be NULL terminated or padded in any way. The length of this field is always 20 octets.
Record Identifier	A 4-byte, unsigned integer containing the Record Identifier value from a software inventory evidence record.
Software Identifier Length	A 2-byte unsigned integer indicating the length in bytes of the Software Identifier field.
Software Locator Length	A 2-byte unsigned integer indicating the length in bytes of the Software Locator field.
Record Len	This is a 4-byte unsigned integer indicating the length of the Record field in bytes.
Action	The type of event that is recorded in this event record. Possible values are: 1 = CREATION - the addition of a record to the endpoint's Software Inventory Evidence Collection; 2 = DELETION - the removal of a record from the endpoint's Software Inventory Evidence Collection; 3 = ALTERATION - There was an alteration to a record within the endpoint's Software Inventory Evidence Collection. All other values are reserved for future use and MUST NOT be used when sending attributes. In the case where a SW-PV receives an event record that uses an action value other than the ones defined here, it MUST ignore that event record but SHOULD process other event records in this attribute as normal.
Data Model Type	A 1-byte unsigned integer containing an identifier number from the Software Data Model IANA table

		that identifies the data model of the reported record.
Software Identifier		A string containing the Software Identifier value from a software inventory evidence record. This field value MUST be normalized to Network Unicode format, as described in Section 4.5. This string MUST NOT be NULL terminated.
Software Locator		A string containing the Software Locator value. This is expressed as a URI. This field value MUST be normalized to Network Unicode format as described in Section 3.2.1.2. This string MUST NOT be NULL terminated.
Record		A software inventory evidence record as a string. The record MUST be converted and normalized to Network Unicode format, as described in Section 4.5. This string MUST NOT be NULL terminated.

Table 8: Software Events Attribute Fields

The fields of this attribute are used in the same way as the corresponding fields of the previous attributes. As with the Software Inventory attribute, a Software Events attribute can be quite large if many events have occurred following the event indicated by a request's Earliest EID. As such, it is recommended that the SW Request attributes only request full records be sent (Result Type set to 0) in a targeted request, thus constraining the response just to records that match a given set of Software Identifiers.

As with the Software Identifier Events Attribute, this attribute MUST only contain event records with EIDs coming from the current EID Epoch of the SW-PC.

As with the Software Inventory Attribute, the SW-PC MUST perform conversion and normalization of the record.

4.12. Subscription Status Request

A SW-PV sends this attribute to a SW-PC to request a list of active subscriptions for which the requesting SW-PV is the subscriber. A SW-PC MUST NOT send this attribute.

This attribute has no fields.

A SW-PC MUST respond to this attribute by sending a Subscription Status Response attribute (or a PA-TNC Error attribute if it is unable to correctly provide a response).

4.13. Subscription Status Response

A SW-PC sends this attribute to a SW-PV to report the list of active subscriptions for which the receiving SW-PV is the subscriber. A SW-PV MUST NOT send this attribute.

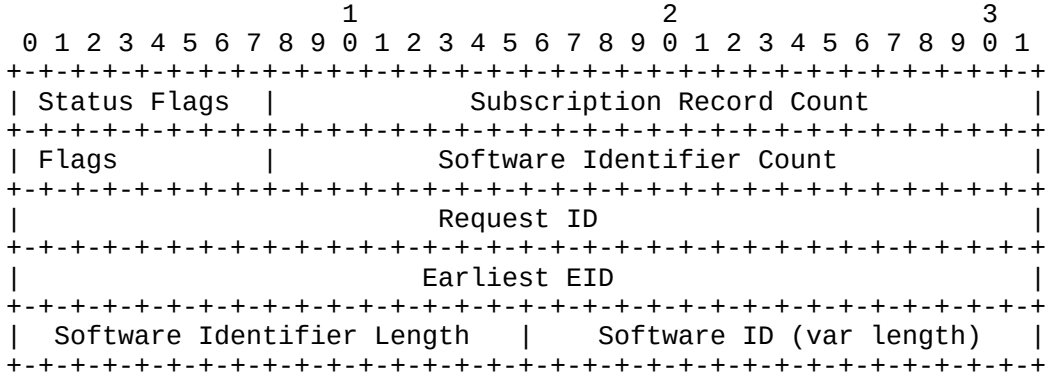


Figure 11: Subscription Status Response Attribute

Field	Description
Flags: Bit 0-7 - Reserved	Reserved for future use. This field MUST be set to zero on transmission and ignored upon reception.
Subscription Record Count	The number of subscription records that follow. This field is a 3-byte unsigned integer. The Flags, Software Identifier Count, Request ID, Earliest EID, Software Identifier Length, and Software ID fields are repeated, in order, the number of times indicated in this field. This field value MAY be 0 in which case there are no instances of these fields.
Flags, Software Identifier Count, Request ID, Earliest EID, Software Identifier Length, and Software ID	For each active subscription, these fields contain an exact copy of the fields with the same name as provided in the subscription's establishing request.

Table 9: Subscription Status Response Fields

A Subscription Status Response contains zero or more subscription records. Specifically, it MUST contain one subscription record for each active subscription associated with the party that sent the Subscription Status Request to which this attribute is a response. As described in Section 3.6.2, the SW-PC MUST use the requester's Connection ID and its Posture Validator ID to determine which subscriptions are associated with the requester.

A SW-PC MUST send a Subscription Status Response attribute in response to a Subscription Status Request attribute. The only exception to this is if the SW-PC experiences an error condition that prevents it from correctly populating the Subscription Status Response attribute, in which case it MUST respond with a PA-TNC Error attribute appropriate to the type of error experienced. If there are no active subscriptions associated with the requesting party, the Subscription Status Response attribute will consist of its Status Flags field, a Subscription Record Count field with a value of 0, and no additional fields.

Each subscription record included in a Subscription Status Response attribute duplicates the fields of a SW Request attribute that was the establishing request of a subscription. Note that the Request ID field in the record captures the Subscription ID associated with the given subscription record (since the Subscription ID is the same as the Request ID of the establishing request). Note also that if the establishing request is targeted, then its Record Count field will be non-zero and, within that subscription record, the Software Identifier Length and Software Identifier fields are repeated, in order, the number of times indicated in the Record Count field. As such, each subscription record can be different sizes. If the establishing request is not targeted (Record Count field is 0), the subscription record has no Software Identifier Length or Software Identifier fields.

When a SW-PV compares the information received in a Subscription Status Response to its own records of active subscriptions it should be aware that the SW-PC might be unable to distinguish this SW-PV from other SW-PVs on the same NEA Server. As a result, it is possible that the SW-PC will report more subscription records than the SW-PV recognizes. For this reason, SW-PVs SHOULD NOT automatically assume that extra subscriptions reported in a Subscription Status Response indicate a problem.

4.14. PA-TNC Error as Used by Software Inventory Message and Attributes for PA-TNC

The PA-TNC Error attribute is defined in the PA-TNC specification [RFC5792], and its use here conforms to that specification. A PA-TNC Error can be sent due to any error in the PA-TNC exchange and might also be sent in response to error conditions specific to the Software Inventory Message and Attributes for PA-TNC exchange. The latter case utilizes error codes defined below.

A PA-TNC Error attribute is sent instead of a SW Response attribute due to factors that prevent the reliable creation of a SW Response. As such, a SW-PC MUST NOT send both a PA-TNC Error attribute and a SW Response attribute in response to a single SW Request attribute.

Table 10 lists the Error Code values for the PA-TNC Error attribute specific to the Software Inventory Message and Attributes for PA-TNC exchange. In all of these cases, the Error Code Vendor ID field MUST be set to 0x000000, corresponding to the IETF SMI Private Enterprise Number. The Error Information structures for each error type are described in the following subsections.

Note that a message with a Software Inventory Message and Attributes for PA-TNC attribute might also result in an error condition covered

by the Standard PA-TNC Error Codes defined in PA-TNC. For example, a SW Attribute might have an invalid parameter, leading to an error code of "Invalid Parameter". In this case, the SW-PC MUST use the appropriate PA-TNC Error Code value as defined in Section 4.2.8 of PA-TNC specification.

Error Code Value	Description
0x00000020	SW_ERROR. This indicates a fatal error (i.e., an error that precludes the creation of a suitable response attribute) other than the errors described below but still specific to the processing of SW Attributes. The Description field SHOULD contain additional diagnostic information.
0x00000021	SW_SUBSCRIPTION_DENIED_ERROR. This indicates that the SW-PC denied the SW-PV's request to establish a subscription. The Description field SHOULD contain additional diagnostic information.
0x00000022	SW_RESPONSE_TOO_LARGE_ERROR. This indicates that the SW-PC's response to the SW-PV's request was too large to be serviced. The error information structure indicates the largest possible size of a response supported by the SW-PC (see Section 4.14.2). The Description field SHOULD contain additional diagnostic information.
0x00000023	SW_SUBSCRIPTION_FULFILLMENT_ERROR. This indicates that the SW-PC experienced an error fulfilling a given subscription. The error information includes the Subscription ID of the relevant subscription, as well as a sub-error that describes the nature of the error the SW-PC experienced. The SW-PC and SW-PV MUST treat the identified subscription as cancelled.
0x00000024	SW_SUBSCRIPTION_ID_REUSE_ERROR. This indicates that the SW-PC received a SW Request from a given SW-PV where the

		Request ID of that SW Request is currently used as the Subscription ID of an active subscription with that SW-PV. This error does not cancel the identified subscription.
0x00000025-0x0000002F		RESERVED. These Error Codes are reserved for use by future revisions of the Software Inventory Message and Attributes for PA-TNC specification. Any PA-TNC Error attribute using one of these Error Codes MUST be treated as indicating a fatal error on the sender without further interpretation.

Table 10: PA-TNC Error Codes for Software Inventory Message and Attributes for PA-TNC

The following subsections describe the structures present in the Error Information fields.

4.14.1. SW_ERROR, SW_SUBSCRIPTION_DENIED_ERROR and SW_SUBSCRIPTION_ID_REUSE_ERROR Information

The SW_ERROR error code indicates that the sender (the SW-PC) has encountered an error related to the processing of a SW Request attribute but which is not covered by more specific SW error codes. The SW_SUBSCRIPTION_DENIED_ERROR is used when the SW-PV requests to establish a subscription or clear all subscriptions from the given SW-PV, but the SW-PC is unable or unwilling to comply with this request. The SW_SUBSCRIPTION_ID_REUSE_ERROR is used when the SW-PC receives a SW Request whose Request ID duplicates a Subscription ID of an active subscription with the request's sender. All of these error codes use the following error information structure.

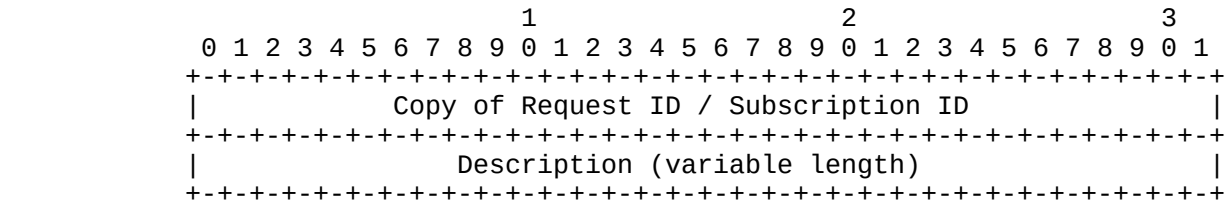


Figure 12: SW Error, Subscription Error, and Subscription ID Reuse Information

Field	Description
Copy of Request ID / Subscription ID	In the case that this error condition is generated in direct response to a SW Request attribute, this field MUST contain an exact copy of the Request ID field in the SW Request attribute that caused this error. In the case that the attribute in question is generated in fulfillment of an active subscription, this field MUST contain the Subscription ID of the subscription for which the attribute was generated. (This is only possible if the error code is SW_ERROR as the other errors are not generated by subscription fulfillment.) Note that, in this case, the indicated error appears as a sub-error for a SW_SUBSCRIPTION_FULFILLMENT_ERROR, as described in Section 4.14.3.
Description	A UTF-8 string describing the condition that caused this error. This field MAY be 0-length. However, senders SHOULD include some description in all PA-TNC Error attributes of these types. This field MUST NOT be NULL terminated.

Table 11: SW Error, Subscription Error, and Subscription ID Reuse Information Fields

This error information structure is used with SW_ERROR, SW_SUBSCRIPTION_DENIED_ERROR, and SW_SUBSCRIPTION_ID_REUSE_ERROR status codes to identify the SW Request attribute that precipitated the error condition and to describe the error. The Description field contains text describing the error. The SW-PC MAY encode machine-interpretable information in this field, but SHOULD also include a human-readable description of the error, since the receiving SW-PV might not recognize the SW-PC’s encoded information.

4.14.2. SW_RESPONSE_TOO_LARGE_ERROR Information

The SW_RESPONSE_TOO_LARGE_ERROR error code indicates that a response generated by a SW-PC in response to a SW-PV’s SW Request attribute was too large to send.

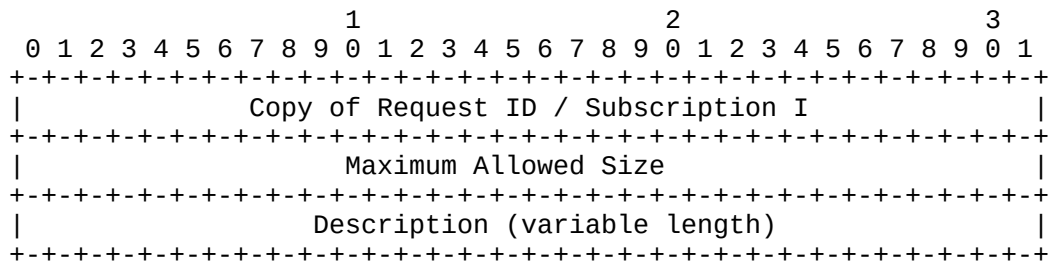


Figure 13: SW Response Too Large Error Information

Field	Description
Copy of Request ID / Subscription ID	In the case that the attribute in question is generated in direct response to a SW Request, this field MUST contain an exact copy of the Request ID field in the SW Request attribute that caused this error. In the case that the attribute in question is generated in fulfillment of an active subscription, this field MUST contain the Subscription ID of the subscription for which the attribute was generated. Note that, in the latter case, the SW_RESPONSE_TOO_LARGE_ERROR appears as a sub-error for a SW_SUBSCRIPTION_FULFILLMENT_ERROR, as described in Section 4.14.3.
Maximum Allowed Size	This field MUST contain an unsigned integer indicating the largest permissible size, in bytes, of SW Attribute that the SW-PC is currently willing to send in response to a SW Request attribute.
Description	A UTF-8 string describing the condition that caused this error. This field MAY be 0-length. However, senders SHOULD include some description in all PA-TNC Error attributes of these types. This field MUST NOT be NULL terminated.

Table 12: SW Response Too Large Error Information Fields

This error structure is used with the SW_RESPONSE_TOO_LARGE_ERROR status code to identify the SW Request attribute that precipitated the error condition and to describe the error. The Maximum Allowed Size field indicates the largest attribute the SW-PC is willing to

send in response to a SW Request under the current circumstances. Note that under other circumstances, the SW-PC might be willing to return larger or smaller responses than indicated (such as if the endpoint connects to the NEA Server using a different network protocol). The other fields in this error information structure have the same meanings as corresponding fields in the SW_ERROR and SW_SUBSCRIPTION_DENIED_ERROR information structure.

4.14.3. SW_SUBSCRIPTION_FULFILLMENT_ERROR Information

The SW_SUBSCRIPTION_FULFILLMENT_ERROR error code indicates that the SW-PC encountered an error while fulfilling a subscription. The bytes after the first 4 octets duplicate a PA-TNC Error attribute (as described in Section 4.2.8 of PA-TNC) that is used to identify the nature of the encountered error.

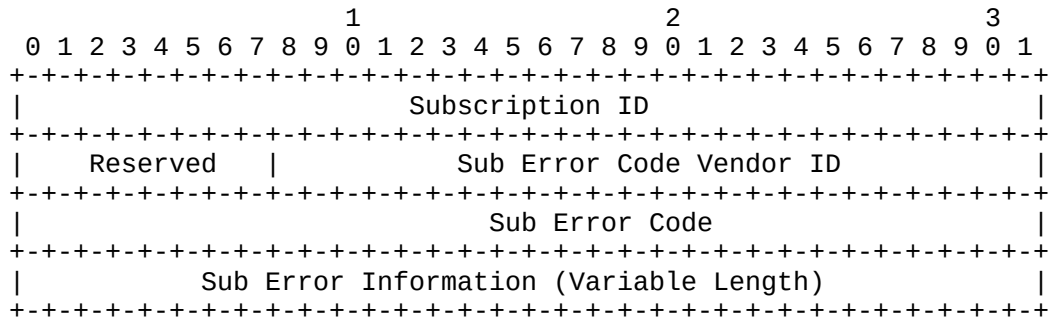


Figure 14: SW Subscription Fulfillment Error Information

Field	Description
Subscription ID	This field MUST contain the Subscription ID of the subscription whose fulfillment caused this error.
Reserved	This field MUST contain the value of the Reserved field of a PA-TNC Error attribute that describes the error condition encountered during subscription processing.
Sub Error Code Vendor ID	This field MUST contain the value of the Error Code Vendor ID field of a PA-TNC Error attribute that describes the error condition encountered during subscription processing.
Sub Error Code	This field MUST contain the value of the Error Code field of a PA-TNC Error attribute that describes the error condition encountered during subscription processing.
Sub Error Information	This field MUST contain the value of the Error Information field of a PA-TNC Error attribute that describes the error condition encountered during subscription processing.

Table 13: SW Subscription Fulfillment Error Information Fields

This error structure is used with the SW_SUBSCRIPTION_FULFILLMENT_ERROR status code. The first 4 octets of this error structure contain the Subscription ID of the subscription that was being fulfilled when the error occurred. The remaining fields of this error structure duplicate the fields of a PA-TNC Error attribute, referred to as the "sub-error". The error code of the sub-error corresponds to the type of error that the SW-PC encountered while fulfilling the given subscription. The sub-error MUST NOT have an error code of SW_SUBSCRIPTION_FULFILLMENT_ERROR.

The SW-PC sending a PA-TNC Error attribute with this error code, and the SW-PV receiving it, MUST treat the subscription identified by the Subscription ID field as cancelled. All other subscriptions are unaffected.

5. Supported Data Models

Software Inventory Message and Attributes for PA-TNC supports an extensible list of data models for representing and transmitting software inventory information. This list of data models appears in the Software Data Model IANA table (see Section 9.4). This document provides guidance for an initial set of data models. Other documents might provide guidance on the use of new data models by Software Inventory Message and Attributes for PA-TNC, and will be referenced by extensions to the Software Data Model IANA table.

5.1. ISO 2015 SWID Tags using XML

The International Organization for Standardization and the International Electrotechnical Commission (ISO/IEC) published the specification governing SWID tag construction and use in 2009 with a revised version published in 2015. [SWID] Since that time, a growing number of vendors have integrated SWID tags into their software products. Doing so significantly simplifies the task of identifying these pieces of software: instead of relying on discovery processes that look for clues as to software presence, such as the presence of particular files or registry keys, a readily available list of SWID tags provides simple and immediate evidence as to the presence of the given piece of software.

SWID Message and Attributes for PA-TNC has no reliance on the presence or management of SWID tags on an endpoint as described in the ISO specification. However, the data model for describing software that is presented in the ISO specification provides a robust and comprehensive way of describing software and is adopted here as a means of representing and transmitting software information. It should be emphasized, the use of the ISO SWID tag data model makes no assumption as to whether the source of the recorded information was, in fact, an ISO SWID tag harvested from the endpoint or whether the information was created using some other source and normalized to the SWID format.

5.1.1. Guidance on Normalizing Source Data to ISO 2015 SWID Tags using XML

TBD

Don’t violate the specification

Use your own Tag Creator RegID or the Unknown Tag Creator RegID. Do not use some other party’s RegID, especially not the RegID of the software author if you are not the author.

5.1.2. Guidance on Creation of Software Identifiers from ISO 2015 SWID Tags

TBD

Use combination of Tag Creator RegID and Unique ID fields. Specifically, format should be NUMBER::TAG_CREATOR_REGID UNIQUE_ID, where NUMBER is the length of TAG_CREATOR_REGID in bytes. The rest of the Software Identifier MUST be the concatenation of the Tag Creator RegID and the Unique ID from the tag, without any connecting character or whitespace.

5.2. ISO 2009 SWID Tags using XML

As noted above, ISO's SWID tag specification provides a useful data model for representation of software information. As of the writing of this specification, while the 2015 specification is considered more comprehensive and addresses some issues with the 2009 specification, 2009-format SWID tags remain far more common in deployments. For this reason, ISO 2009 SWID tags are included in the Software Data Model IANA table.

5.2.1. Guidance on Normalizing Source Data to ISO 2015 SWID Tags using XML

TBD

Don't violate the specification

Use your own Tag Creator RegID or the Unknown Tag Creator RegID. Do not use some other party's RegID, especially not the RegID of the software author if you are not the author.

5.2.2. Guidance on Creation of Software Identifiers from ISO 2015 SWID Tags

TBD

Use combination of Tag Creator RegID and Unique ID fields. Specifically, format should be NUMBER::TAG_CREATOR_REGID UNIQUE_ID, where NUMBER is the length of TAG_CREATOR_REGID in bytes. The rest of the Software Identifier MUST be the concatenation of the Tag Creator RegID and the Unique ID from the tag, without any connecting character or whitespace.

6. Security Considerations

This section discusses some of the security threats facing Posture Collectors and Posture Validators that implement the Software Inventory Message and Attributes for PA-TNC protocol. This section primarily notes potential issues for implementers to consider, although it does contain a handful of normative requirements to address certain security issues. Implementers need to make the final decision as to how their implementations address the given issues. The issues identified below focus on capabilities specific to this document. Implementers are advised to consult other relevant NEA specifications for security issues that are applicable to such components in general.

Reading the Security Considerations section of any well-written specification can be discouraging, as a long list of possible threats is catalogued. Keep in mind that no security measure is absolute, but each one can be beneficial. By understanding the weaknesses of each security measure, we can put in place countermeasures to protect against exploitation of these weaknesses.

6.1. Evidentiary Value of Software Inventory Evidence Records

It must be remembered that the accuracy of an endpoints Software Inventory Evidence Collection as an indicator of the endpoints software load and changes thereon is only as accurate as the tools that populate and manage the software inventory evidence records in this collection. Some tools might not be designed to update records in the Software Inventory Evidence Collection in real time resulting in a collection that is out-of-step with actual system state. Moreover, tools might inaccurately characterize software, or fail to properly record its removal. Finally, it is likely that there will be software on the endpoint that is not tracked by any source and thus is not reflected in the Software Inventory Evidence Collection. Users of collected software inventory evidence records need to understand that the information provided by the Software Inventory Message and Attributes for PA-TNC capability cannot be treated as completely accurate. Nonetheless, having endpoints report this information can still provide useful insights into the state of the endpoint’s software load, and can alert administrators and policy tools of situations that require remediation.

6.2. Sensitivity of Collected Records

Software records on an endpoint are generally not considered to be sensitive, although there can be exceptions to this generalization as noted in the section on Privacy Considerations. In general, an endpoint’s Software Inventory Evidence Collection can be browsed and

individual records read by any party with access to the endpoint. This is generally not considered to be problematic, as those with access to the endpoint can usually learn of everything disclosed by that endpoint's records simply by inspecting other parts of the endpoint.

The situation changes when an endpoint's inventory records are collected and stored off of the endpoint itself, such as on a NEA Server or CMDB. Inventory records represent a wealth of information about the endpoint in question and, for an adversary who does not already have access to the endpoint, a collection of the endpoint's inventory records might provide many details that are useful for mounting an attack. A list of the inventory records associated with an endpoint reveals a list of software installed on the endpoint. This list can be very detailed, noting specific versions and even patch levels, which an adversary can use to identify vulnerable software and design efficacious attacks.

In addition, other information might also be gleaned from a collection of software inventory records:

- An inventory record might include information about where the product was installed on a given endpoint. This can reveal details about the file organization of that endpoint that an attacker can utilize.

- An inventory record might include information about how the software was provided to the endpoint, who in an organization signs off on the package release, and who packaged the product for installation. This information might be used as a starting point for the development of supply chain attacks.

- Events affecting inventory records are reported with timestamps indicating when each given event occurred. This can give the attacker an indication of how quickly an organization distributes patches and updates, helping the attacker determine how long an attack window might remain open.

Any consolidated software inventory is a potential risk, because such an inventory can provide an adversary an insight into the enterprise's configuration and management process. It is recommended that a centralized software inventory record collection be protected against unauthorized access. Mechanisms to accomplish this can include encrypting the data at rest, ensuring that access to the data is limited only to necessary individuals and processes, and other basic security precautions.

6.3. Integrity of Endpoint Records

SW-PCs maintain records of detected changes to the endpoint's Software Inventory Evidence Collection. These records are used to respond to a SW-PV's request for change events. The SW-PV might use a list of reported events to update its understanding of the endpoint's Software Inventory Evidence Collection without needing to receive a full inventory report from the SW-PC. For this reason, preserving the integrity of the SW-PC's record of events is extremely important. If an attacker modifies the SW-PC's record of changes to the endpoint's Software Inventory Evidence Collection, this might cause the SW-PV's understanding of the endpoint's Software Inventory Evidence Collection to differ from its actual state. Results might include leading the SW-PV to believe that absent software was present, that present software was absent, that patches have been installed even if this is not the case, or to be unaware of other changes to Software Inventory Evidence Records. As such, the SW-PC MUST take steps to protect the integrity of its event records.

In addition, records of established SW-PV subscriptions also require protection against manipulation or corruption. If an attacker is able to modify or delete records of an established subscription by a SW-PV, the SW-PC might fail to correctly fulfill this subscription. The SW-PV would not be aware that its subscription was not being correctly fulfilled unless it received additional information that indicated a discrepancy. For example, the SW-PV might collect a full inventory and realize from this that certain events had not been correctly reported in accordance with an established subscription. For this reason, the SW-PC MUST protect the integrity of subscription records.

6.4. SW-PC Access Permissions

A SW-PC requires sufficient permissions to collect Software Inventory Evidence Records from all of its supported sources, as well as sufficient permissions to interact with the endpoint's Posture Broker Client. With regard to the former, this might require permissions to read the contents of directories throughout the file system. Depending on the operating environment and other activities undertaken by a SW-PC (or software that incorporates a SW-PC as one of its capabilities) additional permissions might be required by the SW-PC software. The SW-PC SHOULD NOT be granted permissions beyond what it needs in order to fulfill its duties.

6.5. Sanitization of Record Fields

Not all sources of software inventory evidence are necessarily tightly controlled. For example, consider a source that gathers .swid files from the endpoint's file system. Any party could create a new .swid file that could be collected and turned into a Software Inventory Evidence Record. As a result, it is important that the contents of source information not be automatically trusted. In particular, tools that read source information and the Software Inventory Evidence Records derived therefrom, including SW-PCs, need to be careful to sanitize input to prevent buffer overflow attacks, encoding attacks, and other weaknesses that might be exploited by an adversary who can control the contents of a record.

6.6. PA-TNC Security Threats

In addition to the aforementioned considerations the Software Inventory Message and Attributes for PA-TNC protocol is subject to the same security threats as other PA-TNC transactions, as noted in Section 5.2 of PA-TNC [RFC5792]. These include, but are not limited to, attribute theft, message fabrication, attribute modification, attribute replay, attribute insertion, and denial of service. Implementers are advised to consult the PA-TNC specification to better understand these security issues.

7. Privacy Considerations

As noted in Section 6.2, if an adversary can gain an understanding of the software installed on an endpoint, they can utilize this to launch attacks and maintain footholds on this endpoint. For this reason, the NEA Server needs to ensure adequate safeguards are in place to prevent exposure of collected inventory records. For similar reasons, it is advisable that an endpoint only send records to a NEA Server that is authorized to receive this information and that can be trusted to safeguard this information after collection.

8. Relationship to Other Specifications

This specification makes frequent reference to and use of other specifications. This section describes these relationships.

This specification is expected to participate in a standard NEA architecture. As such, it is expected to be used in conjunction with the other protocols used in a NEA exchange. In particular, SW Attributes are conveyed over PB-TNC [RFC5793], which is in turn conveyed over some variant of PT (either PT-TLS [RFC6876] or PT-EAP [RFC7171]). These protocols have an especially important role, as they are responsible for ensuring that attributes defined under this

specification are delivered reliably, securely, and to the appropriate party.

It is important to note that the Product Information, Numeric Version, and String Version attributes defined in the PA-TNC specification [RFC5792] are also meant to convey information about installed applications and the versions thereof. As such, there is some conceptual overlap between those attributes and the intent of this specification. However, PA-TNC was designed to respond to very specific queries about specific classes of products, while the Software Inventory Message and Attributes for PA-TNC specification is able to convey a broader query, resulting in a more comprehensive set of evidence regarding an endpoint's installed software. As such, this specification provides important capabilities not present in the PA-TNC specification.

9. IANA Considerations

This section extends multiple existing IANA registries. Specifically, it extends the PA-TNC Attribute Types and PA-TNC Error Codes defined in the PA-TNC specification [RFC5792] and the PA-Subtypes registry defined in the PB-TNC specification [RFC5793] and extended in PA-TNC. This specification only adds values to these registries and does not alter how these registries work or are maintained. Consult the appropriate specifications for details on the operations and maintenance of these registries.

9.1. Registry for PA-TNC Attribute Types

Section 4.4 of this specification defines several new PA-TNC attributes. The following values are added to the registry for PA-TNC Attribute Types defined in the PA-TNC specification. Note that Table 3 in Section 4.4 lists these attributes but uses a hexadecimal value to identify their associated integer. The integer values given in that table are identical to those provided here. Note also that Table 3 includes an entry for PA-TNC Error attributes, but the IANA information associated with that attribute is already defined in the PA-TNC specification and is not reproduced here.

PEN	Integer	Name	Defining Specification
0	17	SW Request	Software Inventory Message and Attributes for PA-TNC
0	18	Software Identifier Inventory	Software Inventory Message and Attributes for PA-TNC
0	19	Software Identifier Events	Software Inventory Message and Attributes for PA-TNC
0	20	Software Inventory	Software Inventory Message and Attributes for PA-TNC
0	21	Software Events	Software Inventory Message and Attributes for PA-TNC
0	22	Subscription Status Request	Software Inventory Message and Attributes for PA-TNC
0	23	Subscription Status Response	Software Inventory Message and Attributes for PA-TNC
0	24	Subscription Status Response	Software Inventory Message and Attributes for PA-TNC
0	25 - 31	Reserved for future use	Software Inventory Message and Attributes for PA-TNC

9.2. Registry for PA-TNC Error Codes

Section 4.14 of this specification defines several new PA-TNC Error Codes. The following values are added to the registry for PA-TNC Error Codes defined in the PA-TNC specification. Note that Table 10 in Section 4.14 lists these codes but uses a hexadecimal value to identify their associated integer. The integer values given in that table are identical to those provided here.

PEN	Integer	Name	Defining Specification
0	32	SW_ERROR	Software Inventory Message and Attributes for PA-TNC
0	33	SW_SUBSCRIPTION_DENIED_ERROR	Software Inventory Message and Attributes for PA-TNC
0	34	SW_RESPONSE_TOO_LARGE_ERROR	Software Inventory Message and Attributes for PA-TNC
0	35	SW_SUBSCRIPTION_FULFILLMENT_ERROR	Software Inventory Message and Attributes for PA-TNC
0	36	SW_SUBSCRIPTION_ID_REUSE_ERROR	Software Inventory Message and Attributes for PA-TNC
0	37-47	Reserved for future use	Software Inventory Message and Attributes for PA-TNC

9.3. Registry for PA Subtypes

Section 4.1 of this specification defines one new PA Subtype. The following value is added to the registry for PA Subtypes defined in the PB-TNC specification.

PEN	Integer	Name	Defining Specification
0	9	SW Attributes	Software Inventory Message and Attributes for PA-TNC

9.4. Registry for Software Data Models

TBD

10. References

10.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <http://www.rfc-editor.org/info/rfc2119>.

[RFC3339] Klyne, G. and C. Newman, "Date and Time on the Internet: Timestamps", RFC 3339, DOI 10.17487/RFC3339, July 2002, <http://www.rfc-editor.org/info/rfc3339>.

[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, <http://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, <http://www.rfc-editor.org/info/rfc5198>.

[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, <http://www.rfc-editor.org/info/rfc5209>.

[RFC5792] Sangster, P. and K. Narayan, "PA-TNC: A Posture Attribute (PA) Protocol Compatible with Trusted Network Connect (TNC)", RFC 5792, DOI 10.17487/RFC5792, March 2010, <http://www.rfc-editor.org/info/rfc5792>.

[SWID] The International Organization for Standardization/ International Electrotechnical Commission, "Information Technology - Software Asset Management - Part 2: Software Identification Tag, ISO/IEC 19770-2", November 2009.

10.2. Informative References

[NIST-SWID]
The National Institute of Standards and Technology and The MITRE Corporation, "Guidelines for the Creation of Interoperable Software Identification (SWID) Tags", August 2013.

[RFC5793] Sahita, R., Hanna, S., Hurst, R., and K. Narayan, "PB-TNC: A Posture Broker (PB) Protocol Compatible with Trusted Network Connect (TNC)", RFC 5793, DOI 10.17487/RFC5793, March 2010, <http://www.rfc-editor.org/info/rfc5793>.

[RFC6876] Sangster, P., Cam-Winget, N., and J. Salowey, "A Posture Transport Protocol over TLS (PT-TLS)", RFC 6876, DOI 10.17487/RFC6876, February 2013, <http://www.rfc-editor.org/info/rfc6876>.

[RFC7171] Cam-Winget, N. and P. Sangster, "PT-EAP: Posture Transport (PT) Protocol for Extensible Authentication Protocol (EAP) Tunnel Methods", RFC 7171, DOI 10.17487/RFC7171, May 2014, <http://www.rfc-editor.org/info/rfc7171>.

Authors' Addresses

Chris Coffin
The MITRE Corporation
202 Burlington Road
Bedford, MA 01730
USA

Email: ccoffin@mitre.org

Daniel Haynes
The MITRE Corporation
202 Burlington Road
Bedford, MA 01730
USA

Email: dhaynes@mitre.org

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

Email: cmschmidt@mitre.org

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

Email: jmfitz2@nsa.gov

SACM
Internet-Draft
Intended status: Standards Track
Expires: October 29, 2017

D. Waltermire, Ed.
NIST
K. Watson
DHS
C. Kahn
L. Lorenzin
Pulse Secure, LLC
M. Cokus
D. Haynes
The MITRE Corporation
H. Birkholz
Fraunhofer SIT

April 27, 2017

SACM Information Model
draft-ietf-sacm-information-model-10

Abstract

This document defines the Information Elements that are transported between SACM components and their interconnected relationships. The primary purpose of the Secure Automation and Continuous Monitoring (SACM) Information Model is to ensure the interoperability of corresponding SACM data models and addresses the use cases defined by SACM. The Information Elements and corresponding types are maintained as the IANA "SACM Information Elements" registry.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on October 29, 2017.

Table of Contents

1. Introduction

The SACM Information Model (IM) serves multiple purposes:

- to ensure interoperability between SACM data models that are used as transport encodings,
- to provide a standardized set of Information Elements - the SACM Vocabulary - to enable the exchange of content vital to automated security posture assessment, and
- to enable secure information sharing in a scalable and extensible fashion in order to support the tasks conducted by SACM components.

A complete set of requirements imposed on the IM can be found in [I-D.ietf-sacm-requirements]. The SACM IM is intended to be used for standardized data exchange between SACM components (data in motion). Nevertheless, the Information Elements (IE) and their relationships defined in this document can be leveraged to create and align corresponding data models for data at rest.

The information model expresses, for example, target endpoint (TE) attributes, guidance, and evaluation results. The corresponding Information Elements are consumed and produced by SACM components as they carry out tasks.

The primary tasks that this information model supports (on data, control, and management plane) are:

- TE Discovery
- TE Characterization
- TE Classification

- Collection
- Evaluation
- Information Sharing
- SACM Component Discovery
- SACM Component Authentication
- SACM Component Authorization
- SACM Component Registration

These tasks are defined in [I-D.ietf-sacm-terminology].

2. Conventions used in this document

2.1. Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

2.2. Information Element Examples

The notation used to define the SACM Information Elements (IEs) is based on a customized version of the IPFIX information model syntax [RFC7012] which is described in Figure 2. However, there are several examples presented throughout the document that use a simplified pseudo-code to illustrate the basic structure. It should be noted that while they include actual names of subjects and attributes as well as values, they are not intended to influence how corresponding SACM IEs should be defined in Section 7. The examples are provided for demonstration purposes only.

3. Information Elements

The IEs defined in this document comprise the building blocks by which all SACM content is composed. They are consumed and provided by SACM components on the data plane. Every Information Element has a unique label: its name. Every type of IE defined by the SACM IM is registered as a type at the IANA registry. The Integer Index of the IANA SMI number tables can be used by SACM data models.

3.1. Context of Information Elements

The IEs in this information model represent information related to assets in the following areas (based on the use cases described in [RFC7632]):

- Endpoint Management
- Software Inventory Management
- Hardware Inventory Management
- Configuration Management
- Vulnerability Management

3.2. Extensibility of Information Elements

A SACM data model based on this information model MAY include additional information elements that are not defined here. The labels of additional Information Elements included in different SACM data models MUST NOT conflict with the labels of the Information Elements defined by this information model, and the names of additional Information Elements MUST NOT conflict with each other or across multiple data models. In order to avoid naming conflicts, the labels of additional IEs SHOULD be prefixed to avoid collisions across extensions. The prefix MUST include an organizational identifier and therefore, for example, MAY be an IANA enterprise number, a (partial) name space URI, or an organization name abbreviation.

4. Structure of Information Elements

There are two basic types of IEs:

- Attributes: Atomic information elements that are equivalent to name-value-pairs and can be components of Subjects.
- Subjects: Composite information elements that have a name and are made up of Attributes and/or other Subjects. Every IE that is part of a Subject can have a quantity associated with it (e.g. zero-one, none-unbounded). The content IEs of a Subject can be ordered or unordered.

```
Example Instance of an Attribute:
hostname = "arbutus"

Example Instance of a Subject:
coordinates = (
latitude = N27.99619,
longitude = E86.92761
)
```

Figure 1: Example instance of an attribute and subject.

In general, every piece of information that enables security posture assessment or further enriches the quality of the assessment process can be associated with metadata. In the SACM IM, metadata is represented by specific subjects and is bundled with other attributes or subjects to provide additional information about them. The IM explicitly defines two kinds of metadata:

- Metadata focusing on the data origin (the SACM component that provides the information to the SACM domain)
- Metadata focusing on the data source (the target endpoint that is assessed)

Metadata can also include relationships that refer to other associated IEs (or SACM content in general) by using referencing labels that have to be included in the metadata of the associated IE.

Subjects can be nested and the SACM IM allows for circular or recursive nesting. The association of IEs via nesting results in a tree-like structure wherein subjects compose the root and intermediary nodes and attributes the leaves of the tree. This semantic structure does not impose a specific structure on SACM data models regarding data in motion or data repository schemata for data at rest.

The SACM IM provides two conceptual top-level subjects that are used to ensure a homogeneous structure for SACM content and its associated metadata: SACM statements and SACM content-elements. Every set of IEs that is provided by a SACM component must provide the information contained in these two subjects although it is up to the implementer whether or not the subjects are explicitly defined in a data model.

The notation the SACM IM is defined in is based on a modified version of the IP Information Flow Export (IPFIX) Information Model syntax described in Section 2.1 of [RFC7012]. The customized syntax used by the SACM IM is defined below in Figure 2.

elementId (required): The numeric identifier of the Information Element. It is used for the compact identification of an Information Element. If this identifier is used without an enterpriseID, then the elementId must be unique, and the description of allowed values is administrated by IANA. The value "TBD" may be used during development of the information model until an elementId is assigned by IANA and filled in at publication time.

enterpriseId (optional): Enterprises may wish to define Information Elements without registering them with IANA, for example, for enterprise-internal purposes. For such Information Elements, the elementId is not sufficient when used outside the enterprise. If specifications of enterprise-specific Information Elements are made public and/or if enterprise-specific identifiers are used by SACM components outside the enterprise, then the enterprise-specific identifier MUST be made globally unique by combining it with an enterprise identifier. Valid values for the enterpriseId are defined by IANA as Structure of Management Information (SMI) network management private enterprise numbers.

name (required): A unique and meaningful name for the Information Element.

dataType (required): There are two kinds of datatypes: simple and structured. Attributes are defined using simple datatypes and subjects are defined using structured datatypes. The contents of the datatype field will be either a reference to one of the simple

	datatypes listed in Section 5.1, or the specification of structured datatype as defined in Section 5.2.
status (required):	The status of the specification of the Information Element. Allowed values are "current" and "deprecated". All newly defined Information Elements have "current" status. The process for moving Information Elements to the "deprecated" status is TBD.
description (required):	Describes the meaning of the Information Element, how it is derived, conditions for its use, etc.
structure (optional):	A parsable property that provides details about the definition of structured Information Elements as described in Section 5.2.
references (optional):	Identifies other RFCs or documents outside the IETF which provide additional information or context about the Information Element.

Figure 2: Information Element Specification Template

4.1. Information Element Naming Convention

SACM Information Elements must adhere to the following naming conventions.

- Names SHOULD be descriptive
- Names MUST be unique within the SACM registry. Enterprise-specific names SHOULD be prefixed with a Private Enterprise Number [PEN].
- Names MUST start with lowercase letters unless it begins with a Private Enterprise Number
- Composed names MUST use capital letters for the first letter of each part

4.2. SACM Content Elements

Every piece of information that is provided by a SACM Component is always associated with a set of data source metadata (e.g. the timestamp when the information was collected, the target endpoint from which the this set of information is about, etc.) which is provided in the SACM Content Element Metadata. The SACM Content Element is the subject information element that associates the information with the SACM Content Element Metadata. The SACM Content Element Metadata may also include relationships that express associations with other SACM Content Elements.

```
content-element = (
  content-metadata = (
    collection-timestamp = 146193322,
    data-source = fb02e551-7101-4e68-8dec-1fde6bd10981
  ),
  hostname = "arbutus",
  coordinates = (
  latitude = N27.99619,
  longitude = E86.92761
  )
)
```

Figure 3: Example set of IEs associated with a timestamp and a target endpoint label.

4.3. SACM Statements

One or more SACM Content Elements are bundled in a SACM Statement. In contrast to SACM Content Element Metadata, SACM Statement Metadata focuses on the providing information about the SACM Component that provided it rather than the target endpoint that the content is about. The only content-specific metadata included in the SACM Statement is the statement-type IE. Therefore, multiple SACM Content Elements that share the same SACM Statement Metadata and are of the same statement-type can be included in a single SACM Statement. A SACM Statement functions similar to an envelope or a header and is the subject information element that associates SACM Statement Metadata with security automation information provided in its SACM Content Element(s). Its purpose is to enable the tracking of the origin of data inside a SACM domain and more importantly to enable the mitigation of conflicting information that may originate from different SACM Components. How a consuming SACM Component actually deals with conflicting information is out-of-scope of the SACM IM. Semantically, the term statement implies that the SACM content provided by a SACM Component might not be correct in every context,

but, rather is the result of a best-effort to produce correct information.

```
sacm-statement = (
  statement-metadata = (
    publish-timestamp = 1461934031,
    data-origin = 24e67957-3d31-4878-8892-da2b35e121c2,
    statement-type = observation
  ),
  content-element = (
    content-metadata = (
      collection-timestamp = 146193322,
      data-source = fb02e551-7101-4e68-8dec-1fde6bd10981
    ),
    hostname = "arbutus"
  )
)
```

Figure 4: Example of a simple SACM statement including a single content-element.

```
sacm-statement = (
  statement-metadata = (
    publish-timestamp = 1461934031,
    data-origin = 24e67957-3d31-4878-8892-da2b35e121c2
    statement-type = observation
  ),
  content-element = (
    content-metadata = (
      collection-timestamp = 146193322,
      data-source = fb02e551-7101-4e68-8dec-1fde6bd10981
    ),
    coordinates = (
      latitude = N27.99619,
      longitude = E86.92761
    )
  )
)

sacm-statement = (
  statement-metadata = (
    publish-timestamp = 1461934744,
    data-origin = e42885a1-0270-44e9-bb5c-865cf6bd4800,
    statement-type = observation
  ),
  content-element = (
    content-metadata = (
      collection-timestamp = 146193821,
      te-label = fb02e551-7101-4e68-8dec-1fde6bd10981
    ),
    coordinates = (
      latitude = N16.67622,
      longitude = E141.55321
    )
  )
)
```

Figure 5: Example of conflicting information originating from different SACM components.

4.4. Relationships

An IE can be associated with another IE, e.g. a user-name attribute can be associated with a content-authorization subject. These references are expressed via the relationships subject, which can be included in a corresponding content-metadata subject. The relationships subject includes a list of one or more references. The SACM IM does not enforce a SACM domain to use unique identifiers as

references. Therefore, there are at least two ways to reference another

- The value of a reference represents a specific content-label that is unique in a SACM domain (and has to be included in the corresponding content-element metadata in order to be referenced), or
- The reference is a subject that includes an appropriate number of IEs in order to identify the referenced content-element by its actual content.

It is recommended to provide unique identifiers in a SACM domain and the SACM IM provides a corresponding naming-convention as a reference in Section 4.1. The alternative highlighted above summarizes a valid approach that does not require unique identifiers and is similar to the approach of referencing target endpoints via identifying attributes included in a characterization record.

```
content-element = (
  content-metadata = (
    collection-timestamp = 1461934031,
    te-label =
    fb02e551-7101-4e68-8dec-1fde6bd10981
    relationships = (
      associated-with-user-account =
      f3d70ef4-7e18-42af-a894-8955ba87c95d
    )
  ),
  hostname = "arbutus"
)

content-element = (
  content-metadata = (
    content-label = f3d70ef4-7e18-42af-a894-8955ba87c95d
  ),
  user-account = (
    username = romeo
    authentication = local
  )
)
```

Figure 6: Example instance of a content-element subject associated with another subject via its content metadata.

4.5. Event

Event subjects provide a structure to represent the change of IE values that was detected by a collection task at a specific point of time. It is mandatory to include the new values and the collection timestamp in an event subject and it is recommended to include the past values and a collection timestamp that were replaced by the new IE values. Every event can also be associated with a subject-specific event-timestamp and a lastseen-timestamp that might differ from the corresponding collection-timestamps. If these are omitted the collection-timestamp that is included in the content-metadata subject is used instead.

```
sacm-statement = (
  statement-metadata = (
    publish-timestamp = 1461934031,
    data-origin = 24e67957-3d31-4878-8892-da2b35e121c2,
    statement-type = event
  ),
  event = (
    event-attributes = (
      event-name = "host-name change",
      content-element = (
        content-metadata = (
        collection-timestamp = 146193322,
        data-source =
          fb02e551-7101-4e68-8dec-1fde6bd10981,
          event-component = past-state
       ),
       hostname = "arbutus"
      ),
      content-element = (
        content-metadata = (
          collection-timestamp = 146195723,
          data-source =
          fb02e551-7101-4e68-8dec-1fde6bd10981,
          event-component = current-state
        ),
        hostname = "lilac"
      )
    )
  )
```

Figure 7: Example of a SACM statement containing an event.

4.6. Categories

Categories are special IEs that refer to multiple types of IEs via just one name. Therefore, they are similar to a type-choice. A prominent example of a category is when identifying a target endpoint. In some cases, a target endpoint will be identified by a set of identifying attributes and in other cases a target endpoint will be identified by a target endpoint label which is unique within a SACM domain. If a subject includes the targetEndpoint information element as one of its components, any of the category members (targetEndpointIdentifier or targetEndpointLabel) are valid to be used in its place.

5. Abstract Data Types

This section describes the set of valid abstract data types that can be used for the specification of the SACM Information Elements in Section 7. SACM currently supports two classes of datatypes that can be used to define Information Elements.

- Simple: Datatypes that are atomic and are used to define the type of data represented by an attribute Information Element.
- Structured: Datatypes that can be used to define the type of data represented by a subject Information Element.

Note that further abstract data types may be specified by future extensions of the SACM information model.

5.1. Simple Datatypes

5.1.1. IPFIX Datatypes

To facilitate the use of existing work, SACM supports the following abstract data types defined in Section 3 of [RFC7012].

- unsigned8, unsigned16, unsigned32, unsigned64
- signed8, signed16, signed32, signed64
- float32, float64
- boolean
- macAddress
- octetArray

- string
- dateTimeSeconds, dateTimeMilliseconds, dateTimeMicroseconds, dateTimeNanoSeconds
- ipv4Address, ipv6Address

5.2. Structured Datatypes

5.2.1. List Datatypes

SACM defines the following abstract list data types that are used to represent the structured data associated with subjects.

- list: indicates that the Information Element order is not significant but MAY be preserved.
- orderedList: indicates that Information Element order is significant and MUST be preserved.

The notation for defining a SACM structured datatype is based on regular expressions, which are composed of the keywords "list" or "orderedList" and an Information Element expression. IE expressions use some of the regular expression syntax and operators, but the terms in the expression are the names of defined Information Elements instead of character classes. The syntax for defining list and orderedList datatypes is described below, using BNF:

```
<list-def> -> ("list"|"orderedList") "(" <ie-expression> ")"

<ie-expression> -> <ie-name> <cardinality>?
                   ( ("," | "|") <ie-name> <cardinality>?)*

<cardinality> -> "*" | "+" | "?" |
                 ( "(" <non-neg-int> ("," <non-neg-int>)? ")" )
```

Figure 8: Syntax for Defining List Datatypes

As seen above, multiple occurrences of an Information Element may be present in a structured datatype. The cardinality of an Information Element within a structured Information Element definition is defined by the following operators:

```
       * - zero or more occurrences

       + - one or more occurrences

       ? - zero or one occurrence

      (m,n) - between m and n occurrences
```

Figure 9: Specifying Cardinality for Structured Datatypes

The absence of a cardinality operator implies one mandatory occurrence of the Information Element.

Below is an example of a structured Information Element definition.

```
personInfo = list(firstName, middleNames?, lastName)
firstName = string
middleNames = orderedList(middleName+)
middleName = string
lastName = string
```

As an example, consider the name "John Ronald Reuel Tolkien". Below are instances of this name, structured according to the personInfo definition.

```
personInfo = (firstName="John", middleNames(middleName="Ronald",
              middleName="Reuel"), lastName="Tolkien")

personInfo = (middleNames(middleName="Ronald", middleName=" Reuel"),
              lastName="Tolkien", firstName="John")
```

The instance below is not legal with respect to the definition of personInfo because the order in middleNames is not preserved.

```
personInfo = (firstName="John", middleNames(middleName=" Reuel",
              middleName="Ronald"), lastName="Tolkien")
```

Figure 10: Example of Defining a Structured List Datatype

5.2.2. Enumeration Datatype

SACM defines the following abstract enumeration datatype that is used to represent the restriction of an attribute value to a set of values.

```
name, hex-value, description
<enumeration-def> -> -> <name> ";" <hex-value> ";" <description>
<name> -> [0-9a-zA-Z]+
<hex-value> -> 0x[0-9a-fA-F]+
<description> -> [0-9a-zA-Z\.\,]+
```

Figure 11: Syntax for Defining an Enumeration Datatype

Below is an example of a structured Information Element definition for an enumeration.

```
                          Red    ; 0x1  ; The color is red.
                          Orange ; 0x2  ; The color is orange.
                          Yellow ; 0x3  ; The color is yellow.
                          Green  ; 0x4  ; The color is green.
                          ...
```

Figure 12: Example of Defining a Structured Enumeration Datatype

5.2.3. Category Datatype

SACM defines the following abstract category datatype that is used to represent a type-choice between a set of information elements.

```
<category-def> -> "category(" <ie-expression> ")"
<ie-expression> -> <ie-name> ("|" <ie-name>)*
<name> -> [0-9a-zA-Z]+
```

Figure 13: Syntax for Defining an Category Datatype

Below is an example of a structured Information Element definition for a category.

```
targetEndpoint = category(targetEndpointIdentifier |
                          targetEndpointLabel)
```

Figure 14: Example of Defining a Structured Category Datatype

6. Information Model Assets

In order to represent the Information Elements related to the areas listed in Section 3.1, the information model defines the information needs (or metadata about those information needs) related to following types of assets which are defined in [I-D.ietf-sacm-terminology] (and included below for convenience) which are of interest to SACM. Specifically:

- Endpoint

- Software Component
- Hardware Component
- Identity
- Guidance
- Evaluation Results

The following figure shows the make up of an Endpoint asset which contains zero or more hardware components and zero or more software components each of which may have zero or more instances running an endpoint at any given time as well as zero or more identities that act on behalf of the endpoint when interfacing with other endpoints, tools, or services. An endpoint may also contain other endpoints in the case of a virtualized environment.

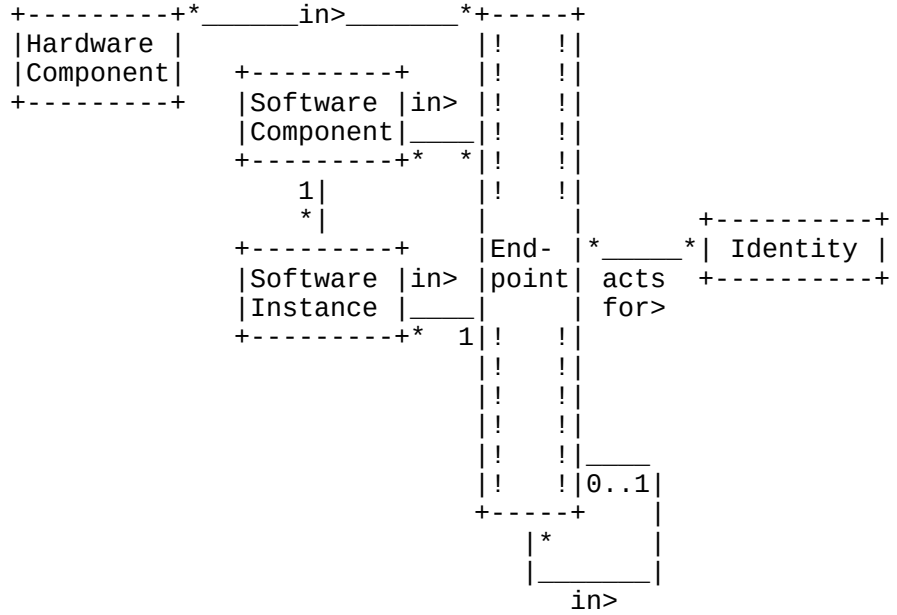


Figure 15: Model of an Endpoint

6.1. Asset

As defined in [RFC4949], an asset is a system resource that is (a) required to be protected by an information system's security policy,

(b) intended to be protected by a countermeasure, or (c) required for a system's mission.

In the scope of SACM, an asset can be composed of other assets. Examples of Assets include: Endpoints, Software, Guidance, or Identity. Furthermore, an asset is not necessarily owned by an organization.

6.2. Endpoint

From [RFC5209], an endpoint is any computing device that can be connected to a network. 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, 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).

6.3. Hardware Component

Hardware components are the distinguishable physical components that compose an endpoint. The composition of an endpoint can be changed over time by adding or removing hardware components. In essence, every physical endpoint is potentially a composite of multiple hardware components, typically resulting in a hierarchical composition of hardware components. The composition of hardware components is based on interconnects provided by specific hardware types (e.g. mainboard is a hardware type that provides local busses as an interconnect). In general, a hardware component can be distinguished by its serial number.

Examples of a hardware components include: motherboards, network interfaces, graphics cards, hard drives, etc.

6.4. Software Component

A software package installed on an endpoint (including the operating system) as well as a unique serial number if present (e.g. a text editor associated with a unique license key).

It should be noted that this includes both benign and harmful software packages. Examples of benign software components include: applications, patches, operating system kernel, boot loader, firmware, code embedded on a webpage, etc. Examples of malicious software components include: malware, trojans, viruses, etc.

6.4.1. Software Instance

A running instance of the software component (e.g. on a multi-user system, one logged-in user has one instance of a text editor running and another logged-in user has another instance of the same text editor running, or on a single-user system, a user could have multiple independent instances of the same text editor running).

6.5. Identity

Any mechanism that can be used to identify an asset during an authentication process. Examples include usernames, user and device certificates, etc. Note, that this is different than the identity of assets in the context of designation as described in Section 11.1.

6.6. Guidance

Guidance is input instructions to processes and tasks, such as collection or evaluation. Guidance influences the behavior of a SACM component and is considered content of the management plane. Guidance can be manually or automatically generated or provided. Typically, the tasks that provide guidance to SACM components have a low-frequency and tend to be sporadic. A prominent example of guidance are target endpoint profiles,but guidance can have many forms, including:

Configuration, e.g. a SACM component’s name, or a CMDB’s IPv6 address.

Profiles, e.g. a set of expected states for network behavior associated with target endpoints employed by specific users.

Policies, e.g. an interval to refresh the registration of a SACM component, or a list of required capabilities for SACM components in a specific location.

6.6.1. Collection Guidance

A collector may need guidance to govern what it collects and when. Collection Guidance provides instructions for a Collector that specifies which endpoint attributes to collect, when to collect them, and how to collect them. Collection Guidance is composed of Target Endpoint Attribute Guidance, Frequency Guidance, and Method Guidance.

- Target Endpoint Attribute Guidance: Set of endpoint attributes that are supposed to be collected from a target endpoint. The definition of the set of endpoint attributes is typically based on an endpoint characterization record.
- Frequency Guidance: Specifies when endpoint attributes are to be collected.
- Method Guidance: Indicates how endpoint attributes are to be collected.

6.6.2. Evaluation Guidance

An evaluator typically needs guidance to govern what it considers to be a good or bad security posture. Evaluation Guidance provides instructions for an Evaluator that specifies which endpoint attributes to evaluate, the desired state of those endpoint attributes, and any special requirements that enable an Evaluator to determine if the endpoint attributes can be used in the evaluation (e.g. freshness of data, how it was collected, etc.). Evaluation Guidance is composed of Target Endpoint Attribute Guidance, Expected Endpoint Attribute Value Guidance, and Frequency Guidance.

- Target Endpoint Attribute Guidance: Set of target endpoint attributes that are supposed to be used in an evaluation as well as any requirements on the endpoint attributes. The definition of the set of endpoint attributes is typically based on an endpoint characterization record.
- Expected Endpoint Attribute Value Guidance: The expected values of the endpoint attributes described in the Target Endpoint Attribute Guidance.
- Frequency Guidance: Specifies when endpoint attributes are to be evaluated.
- Method Guidance: Indicates how endpoint attributes are to be collected.

6.6.3. Classification Guidance

A SACM Component carrying out the Target Endpoint Classification Task may need guidance on how to classify an endpoint. Specifically, how to associate endpoint classes with a specific target endpoint characterization record. Target Endpoint Classes function as guidance for collection, evaluation, remediation and security posture assessment in general. Classification Guidance is composed of Target Endpoint Attribute Guidance and Class Guidance.

- Target Endpoint Attribute Guidance: Set of target endpoint attributes that are supposed to be used to identify the endpoint characterization record.

- Class Guidance: A list of target endpoint classes that are to be associated with the identified target endpoint characterization record.

6.6.4. Storage Guidance

An SACM Component typically needs guidance to govern what information it should store and where. Storage Guidance provides instructions for a SACM Component that specifies which security automation information should be stored, for how long, and on which endpoint. Storage Guidance is composed of Target Endpoint Attribute Guidance, Expected Security Automation Information Guidance, and Retention Guidance.

- Target Endpoint Attribute Guidance: Set of target endpoint attributes that are supposed to be used to identify the endpoint where the security automation information is to be stored.

- Expected Security Automation Information Guidance: The security automation information that is expected to be stored (guidance, collected posture attributes, results, etc.).

- Retention Guidance: Specifies how long the security automation information should be stored.

6.6.5. Evaluation Results

Evaluation Results are the output of comparing the actual state of an endpoint against the expected state of an endpoint. In addition to the actual results of the comparison, Evaluation Results should include the Evaluation Guidance and actual target endpoint attributes values used to perform the evaluation.

7. Information Model Elements

This section defines the specific Information Elements and relationships that will be implemented by data models and transported between SACM Components.

7.1. sacmStatement

```
elementId: TBD
name: sacmStatement
dataType: orderedList
status: current
description: Associates SACM Statement Metadata
which provides data origin information about
the providing SACM Component with one or more
SACM Content Elements that contain security
automation information.
structure: orderedList(sacmStatementMetadata,
                       sacmContentElement+)
```

7.2. sacmStatementMetadata

```
elementId: TBD
name: sacmStatementMetadata
dataType: orderedList
status: current
description: Contains IEs that provide
information about the data origin of the
providing SACM Component as well as the
information necessary for other SACM
Components to understand the type of
security automation information in the
SACM Statement's SACM Content Element(s).
structure: orderedList(publicationTimestamp,
                       dataOrigin, anyIE*)
```

7.3. sacmContentElement

```
elementId: TBD
name: sacmContentElement
dataType: list
status: current
description: Associates SACM Content Element
Metadata which provides information about the
data source and type of security automation
information with the actual security automation
information.
structure: TODO
```

7.4. sacmContentElementMetadata

```
elementId: TBD
name: sacmContentElementMetadata
dataType: orderedList
status: current
description: Contains IEs that provide
information about the data source and type of
security automation information such that other
SACM Components are able to parse and understand
the security automation information contained
within the SACM Statement's SACM Content Element(s).
structure: orderedList(collectionTimestamp,
                       targetEndpoint, anyIE*)
```

7.5. targetEndpoint

```
elementId: TBD
name: targetEndpoint
dataType: category
status: current
description: Information that identifies a target
        endpoint on the network. This may be a set of
attributes that can be used to identify an endpoint
on the network or a label that is unique to a SACM
domain.
        structure: category(targetEndpointIdentifier |
                            targetEndpointLabel)
```

7.6. targetEndpointIdentifier

```
elementId: TBD
name: targetEndpointIdentifier
dataType: list
status: current
description: A set of attributes that uniquely
             identify a target endpoint on the network.
structure: list(anyIE+)
```

7.7. targetEndpointLabel

```
elementId: TBD
name: targetEndpointLabel
dataType: string
status: current
description: A label that uniquely identifies
             a target endpoint on SACM domain.
```

7.8. anyIE

elementId: TBD
name: anyIE
dataType: category
status: current
description: This category is a placeholder
for any information element defined within
the SACM Information Model. Its purpose is
to provide an extension point in other
information elements that enable them to
support the specific needs of an enterprise,
user, product, or service.

7.9. accessPrivilegeType

elementId: TBD
name: accessPrivilegeType
dataType: string
status: current
description: A set of types that represent access
privileges (read, write, none, etc.).

7.10. accountName

elementId: TBD
name: accountName
dataType: string
status: current
description: A label that uniquely identifies an account
that can require some form of (user) authentication to
access.

7.11. administrativeDomainType

elementId: TBD
name: administrativeDomainType
dataType: string
status: current
description: A label the is supposed to uniquely
identify an administrative domain.

7.12. addressAssociationType

```
elementId: TBD
name: addressAssociationType
dataType: string
status: current
description: A label the is supposed to uniquely
identify an administrative domain.
```

7.13. addressMaskValue

```
elementId: TBD
name: addressMaskValue
dataType: string
status: current
description: A value that expresses a generic address
subnetting bitmask.
```

7.14. addressType

```
elementId: TBD
name: addressType
dataType: string
status: current
description: A set of types that specifies the type
of address that is expressed in an address subject
(e.g. ethernet, modbus, zigbee).
```

7.15. addressValue

```
elementId: TBD
name: addressValue
dataType: string
status: current
description: A value that expresses a generic network
             address.
```

7.16. applicationComponent

```
elementId: TBD
name: applicationComponent
dataType: string
status: current
description: A label that references a "sub"-application
that is part of the application (e.g. an add-on, a
cipher-suite, a library).
```

7.17. applicationLabel

elementId: TBD
name: applicationLabel
dataType: string
status: current
description: A label that is supposed to uniquely reference an application.

7.18. applicationType

elementId: TBD
name: applicationType
dataType: string
status: current
description: A set of types (FIXME maybe a finite set is not realistic here - value not enumerator?) that identifies the type of (user-space) application (e.g. text-editor, policy-editor, service-client, service-server, calendar, rouge-like RPG).

7.19. applicationManufacturer

elementId: TBD
name: applicationManufacturer
dataType: string
status: current
description: The name of the vendor that created the application.

7.20. authenticator

elementId: TBD
name: authenticator
dataType: string
status: current
description: A label that references a SACM component that can authenticate target endpoints (can be used in a target-endpoint subject to express that the target endpoint was authenticated by that SACM component.

7.21. authenticationType

```
elementId: TBD
name: authenticationType
dataType: string
status: current
description: A set of types that express which type
of authentication was used to enable a network
interaction/connection.
```

7.22. birthdate

```
elementId: TBD
name: birthdate
dataType: string
status: current
description: A label for the registered day of
birth of a natural person (e.g. the date of birth
of a person as an ISO date string).
references: http://rs.tdwg.org/ontology/voc/Person#birthdate
```

7.23. bytesReceived

```
elementId: TBD
name: bytesReceived
dataType: string
status: current
description: A value that represents a number of octets
received on a network interface.
```

7.24. bytesReceived

```
elementId: TBD
name: bytesReceived
dataType: string
status: current
description: A value that represents the number of
octets received on a network interface.
```

7.25. bytesSent

```
elementId: TBD
name: bytesSent
dataType: string
status: current
description: A value that represents the number of
octets sent on a network interface.
```

7.26. certificate

elementId: TBD
name: certificate
dataType: string
status: current
description: A value that expresses a certificate that can be collected from a target endpoint.

7.27. collectionTaskType

elementId: TBD
name: collectionTaskType
dataType: string
status: current
description: A set of types that defines how collected SACM content was acquired (e.g. network-observation, remote-acquisition, self-reported, derived, authority, verified).

7.28. confidence

elementId: TBD
name: confidence
dataType: string
status: current
description: A representation of the subjective probability that the assessed value is correct. If no confidence value is given, it is assumed that the confidence is 1. Acceptable values are between 0 and 1.

7.29. contentAction

elementId: TBD
name: contentAction
dataType: string
status: current
description: A set of types that express a type of action (e.g. add, delete, update). It can be associated, for instance, with an event subject or with a network observation.

7.30. countryCode

elementId: TBD
name: countryCode
dataType: string
status: current
description: A set of types according to ISO 3166-1.

7.31. dataOrigin

elementId: TBD
name: dataOrigin
dataType: string
status: current
description: A label that uniquely identifies a SACM component in and across SACM domains.

7.32. dataSource

elementId: TBD
name: dataSource
dataType: string
status: current
description: A label that is supposed to uniquely identify the data source (e.g. a target endpoint or sensor) that provided an initial endpoint attribute record.

7.33. default-depth

elementId: TBD
name: default-depth
dataType: string
status: current
description: A value that expresses how often a circular reference of subject is allowed to repeat, or how deep a recursive nesting may occur, respectively.

7.34. discoverer

elementId: TBD
name: discoverer
dataType: string
status: current
description: A label that refers to the SACM component that discovered a target endpoint (can be used in a target-endpoint subject to express, for example, that the target endpoint was authenticated by that SACM component).

7.35. emailAddress

elementId: TBD
name: emailAddress
dataType: string
status: current
description: A value that expresses an email-address.

7.36. eventType

elementId: TBD
name: eventType
dataType: string
status: current
description: a set of types that define the categories of an event (e.g. access-level-change, change-of-privilege, change-of-authorization, environmental-event, or provisioning-event).

7.37. eventThreshold

elementId: TBD
name: eventThreshold
dataType: string
status: current
description: If applicable, a value that can be included in an event subject to indicate what numeric threshold value was crossed to trigger that event.

7.38. eventThresholdName

elementId: TBD
name: eventThresholdName
dataType: string
status: current
description: If an event is created due to a crossed threshold, the threshold might have a name associated with it that can be expressed via this value.

7.39. eventTrigger

elementId: TBD
name: eventTrigger
dataType: string
status: current
description: This value is used to express more complex trigger conditions that may cause the creation of an event.

7.40. firmwareId

```
elementId: TBD
name: firmwareId
dataType: string
status: current
description: A label that represents the BIOS or
firmware ID of a specific target endpoint.
```

7.41. hostName

```
elementId: TBD
name: hostName
dataType: string
status: current
description: A label typically associated with an
endpoint, but, not always intended to be unique given
scope.
```

7.42. interfaceLabel

```
elementId: TBD
name: interfaceLabel
dataType: string
status: current
description: A unique label that can be used to
             reference a network interface.
```

7.43. ipv6AddressSubnetMask

```
elementId: TBD
name: ipv6AddressSubnetMask
dataType: string
status: current
description: An IPv6 subnet bitmask.
```

7.44. ipv6AddressSubnetMaskCidrNotation

```
elementId: TBD
name: ipv6AddressSubnetMaskCidrNotation
dataType: string
status: current
description: An IPv6 subnet bitmask in CIDR notation.
```

7.45. ipv6AddressValue

```
elementId: TBD
name: ipv6AddressValue
dataType: ipv6Address
status: current
description: An IPv6 subnet bitmask in CIDR notation.
```

7.46. ipv4AddressSubnetMask

```
elementId: TBD
name: ipv4AddressSubnetMask
dataType: string
status: current
description: An IPv4 subnet bitmask.
```

7.47. ipv4AddressSubnetMaskCidrNotation

```
elementId: TBD
name: ipv4AddressSubnetMaskCidrNotation
dataType: string
status: current
description: An IPv4 subnet bitmask in CIDR notation.
```

7.48. ipv4AddressValue

```
elementId: TBD
name: ipv4AddressValue
dataType: ipv4Address
status: current
description: An IPv4 address value.
```

7.49. layer2InterfaceType

```
elementId: TBD
name: layer2InterfaceType
dataType: string
status: current
description: A set of types referenced by IANA ifType.
```

7.50. layer4PortAddress

```
elementId: TBD
name: layer4PortAddress
dataType: unsigned32
status: current
description: A layer 4 port address
typically associated with TCP and UDP
protocols.
```

7.51. layer4Protocol

```
elementId: TBD
name: layer4Protocol
dataType: string
status: current
description: A set of types that express a layer 4
protocol (e.g. UDP or TCP).
```

7.52. locationName

```
elementId: TBD
name: locationName
dataType: string
status: current
description: A value that represents a named region of
             physical space.
```

7.53. networkZoneLocation

```
elementId: TBD
name: networkZoneLocation
dataType: string
status: current
description: The zone location of an endpoint on the
             network (e.g. internet, enterprise DMZ,
             enterprise WAN, enclave DMZ, enclave).
```

7.54. layer2NetworkLocation

```
elementId: TBD
name: layer2NetworkLocation
dataType: string
status: current
description: The location of a layer-2 interface on
the network (e.g. link-layer neighborhood,
shared broadcast domain).
```

7.55. layer3NetworkLocation

elementId: TBD
name: layer3NetworkLocation
dataType: string
status: current
description: The location of a layer-3 interface on the network (e.g. next-hop routing neighbor).

7.56. macAddressValue

elementId: TBD
name: macAddressValue
dataType: string
status: current
description: A value that expresses an Ethernet address.

7.57. methodLabel

elementId: TBD
name: methodLabel
dataType: string
status: current
description: A label that references a specific method registered and used in a SACM domain (e.g. method to match and re-identify target endpoints via identifying attributes).

7.58. methodRepository

elementId: TBD
name: methodRepository
dataType: string
status: current
description: A label that references a SACM component methods can be registered at and that can provide guidance in the form of registered methods to other SACM components.

7.59. networkAccessLevelType

elementId: TBD
name: networkAccessLevelType
dataType: string
status: current
description: A set of types that express categories of network access-levels (e.g. block, quarantine, etc.).

7.60. networkId

```
elementId: TBD
name: networkId
dataType: string
status: current
description: Most networks such as AS, OSBF domains,
or VLANs can have an ID.
```

7.61. networkInterfaceName

```
elementId: TBD
name: networkInterfaceName
dataType: string
status: current
description: A label that uniquely identifies an
interface associated with a distinguishable endpoint.
```

7.62. networkLayer

```
elementId: TBD
name: networkLayer
dataType: string
status: current
description: A set of layers that expresses the specific
network layer an interface operates on.
```

7.63. networkName

```
elementId: TBD
name: networkName
dataType: string
status: current
description: A label that is associated with a network.
Some networks, for example, effective
layer2-broadcast-domains are difficult to "grasp" and
therefore quite difficult to name.
```

7.64. organizationId

```
elementId: TBD
name: organizationId
dataType: string
status: current
description: A label that uniquely identifies an
             organization via a PEN.
```

7.65. patchId

elementId: TBD
name: patchId
dataType: string
status: current
description: A label that uniquely identifies a specific software patch.

7.66. patchName

elementId: TBD
name: patchName
dataType: string
status: current
description: The vendor’s name of a software patch.

7.67. personFirstName

elementId: TBD
name: personFirstName
dataType: string
status: current
description: The first name of a natural person.

7.68. personLastName

elementId: TBD
name: personLastName
dataType: string
status: current
description: The last name of a natural person.

7.69. personMiddleName

elementId: TBD
name: personMiddleName
dataType: string
status: current
description: The middle name of a natural person.

7.70. phoneNumber

```
elementId: TBD
name: phoneNumber
dataType: string
status: current
description: A label that expresses the U.S. national
phone number (e.g. pattern value="((\d{3}) )?\d{3}-\d{4}").
```

7.71. phoneNumberType

```
elementId: TBD
name: phoneNumberType
dataType: string
status: current
description: A set of types that express the type of
a phone number (e.g. DSN, Fax, Home, Mobile, Pager,
Secure, Unsecure, Work, Other).
```

7.72. privilegeName

```
elementId: TBD
name: privilegeName
dataType: string
status: current
description: The attribute name of the privilege
represented as an AVP.
```

7.73. privilegeValue

```
elementId: TBD
name: privilegeValue
dataType: string
status: current
description: The value content of the privilege
represented as an AVP.
```

7.74. protocol

```
elementId: TBD
name: protocol
dataType: string
status: current
description: A set of types that defines specific
protocols above layer 4 (e.g. http, https, dns, ipp,
or unknown).
```

7.75. publicKey

elementId: TBD
name: publicKey
dataType: string
status: current
description: The value of a public key (regardless of its method of creation, crypto-system, or signature scheme) that can be collected from a target endpoint.

7.76. relationshipContentElementGuid

elementId: TBD
name: relationshipContentElementGuid
dataType: string
status: current
description: A reference to a specific content element used in a relationship subject.

7.77. relationshipStatementElementGuid

elementId: TBD
name: relationshipStatementElementGuid
dataType: string
status: current
description: A reference to a specific SACM statement used in a relationship subject.

7.78. relationshipObjectLabel

elementId: TBD
name: relationshipObjectLabel
dataType: string
status: current
description: A reference to a specific label used in content (e.g. a te-label or a user-id). This reference is typically used if matching content attribute can be done efficiantly and can also be included in addition to a relationship-content-element-guid reference.

7.79. relationshipType

elementId: TBD
name: relationshipType
dataType: string
status: current
description: A set of types that is in every instance of a relationship subject to highlight what kind of relationship exists between the subject the relationship is included in (e.g. associated_with_user, applies_to_session, seen_on_interface, associated_with_flow, contains_virtual_device).

7.80. roleName

elementId: TBD
name: roleName
dataType: string
status: current
description: A label that references a collection of privileges assigned to a specific entity.

7.81. sessionStateType

elementId: TBD
name: sessionStateType
dataType: string
status: current
description: A set of types a discernible session (an ongoing network interaction) can be in (e.g. Authenticating, Authenticated, Postured, Started, Disconnected).

7.82. statementGuid

elementId: TBD
name: statementGuid
dataType: string
status: current
description: A label that expresses a global unique ID referencing a specific SACM statement that was produced by a SACM component.

7.83. statementType

```
elementId: TBD
name: statementType
dataType: string
status: current
description: A set of types that define the type of
content that is included in a SACM statement (e.g.
Observation, DirectoryContent, Correlation, Assessment,
Guidance, Event).
```

7.84. status

```
elementId: TBD
name: status
dataType: string
status: current
description: A set of types that defines possible
result values for a finding in general (e.g. true,
false, error, unknown, not applicable, not evaluated).
```

7.85. subAdministrativeDomain

```
elementId: TBD
name: subAdministrativeDomain
dataType: string
status: current
description: A label for related child domains an
administrative domain can be composed of (used in the
subject administrativeDomain).
```

7.86. subInterfaceLabel

```
elementId: TBD
name: subInterfaceLabel
dataType: string
status: current
description: A unique label a sub network interface
(e.g. a tagged vlan on a trunk) can be referenced
with.
```

7.87. superAdministrativeDomain

```
elementId: TBD
name: superAdministrativeDomain
dataType: string
status: current
description: a label for related parent domains an
             administrative domain is part of (used
             in the subject administrativeDomain).
```

7.88. superInterfaceLabel

```
elementId: TBD
name: superInterfaceLabel
dataType: string
status: current
description: a unique label a super network interface
             (e.g. a physical interface a tunnel
             interface terminates on) can be referenced
             with.
```

7.89. teAssessmentState

```
elementId: TBD
name: teAssessmentState
dataType: string
status: current
description: a set of types that defines the state of
             assessment of a target-endpoint (e.g.
             in-discovery, discovered, in-classification,
             classified, in-assessment, assessed).
```

7.90. teLabel

```
elementId: TBD
name: teLabel
dataType: string
status: current
description: an identifying label created from a set
             of identifying attributes used to reference
             a specific target endpoint.
```

7.91. teId

```
elementId: TBD
name: teId
dataType: string
status: current
description: an identifying label that is created
             randomly, is supposed to be unique, and
             used to reference a specific target
             endpoint.
```

7.92. timestampType

```
elementId: TBD
name: timestampType
dataType: string
status: current
description: a set of types that express what type of
             action or event happened at that point
             of time (e.g. discovered, classified,
             collected, published). Can be included in
             a generic timestamp subject.
```

7.93. unitsReceived

```
elementId: TBD
name: unitsReceived
dataType: string
status: current
description: a value that represents a number of units
             (e.g. frames, packets, cells or segments)
             received on a network interface.
```

7.94. unitsSent

```
elementId: TBD
name: unitsSent
dataType: string
status: current
description: a value that represents a number of units
             (e.g. frames, packets, cells or segments)
             sent on a network interface.
```

7.95. userDirectory

```
elementId: TBD
name: userDirectory
dataType: string
status: current
description: a label that identifies a specific type
of user-directory (e.g. ldap, active-directory,
local-user).
```

7.96. sacmUserId

```
elementId: TBD
name: sacmUserId
dataType: string
status: current
description: a label that references a specific user
known in a SACM domain.
```

7.97. webSite

```
elementId: TBD
name: webSite
dataType: string
status: current
description: a URI that references a web-site.
```

7.98. WGS84Longitude

```
elementId: TBD
name: WGS84Longitude
dataType: float64
status: current
description: a label that represents WGS 84 rev 2004
longitude.
```

7.99. WGS84Latitude

```
elementId: TBD
name: WGS84Latitude
dataType: float64
status: current
description: a label that represents WGS 84 rev 2004
latitude.
```

7.100. WGS84Altitude

```
elementId: TBD
name: WGS84Altitude
dataType: float64
status: current
description: a label that represents WGS 84 rev 2004
altitude.
```

7.101. hardwareSerialNumber

```
elementId: TBD
name: hardwareSerialNumber
dataType: string
status: current
description: A globally unique identifier for a
             particular piece of hardware assigned
             by the vendor.
```

7.102. interfaceName

```
elementId: TBD
name: interfaceName
dataType: string
status: current
description: A short name uniquely describing an
             interface, e.g. "Eth1/0". See [RFC2863]
             for the definition of the ifName object.
```

7.103. interfaceIndex

```
elementId: TBD
name: interfaceIndex
dataType: unsigned32
status: current
description: The index of an interface installed on an endpoint.
             The value matches the value of managed object
             'ifIndex' as defined in [RFC2863]. Note that ifIndex
             values are not assigned statically to an interface
             and that the interfaces may be renumbered every time
             the device's management system is re-initialized,
             as specified in [RFC2863].
```

7.104. interfaceMacAddress

```
elementId: TBD
name: interfaceMacAddress
dataType: macAddress
status: current
description: The IEEE 802 MAC address associated with a network
             interface on an endpoint.
```

7.105. interfaceType

```
elementId: TBD
name: interfaceType
dataType: unsigned32
status: current
description: The type of a network interface. The value matches
             the value of managed object 'ifType' as defined in
             [IANA registry ianaiftype-mib].
```

7.106. interfaceFlags

```
   elementId: TBD
   name: interfaceFlags
   dataType: unsigned16
   status: current
   description: This information element specifies the flags
                associated with a network interface. Possible
                values include:
   structure:
          Up                  ; 0x1   ; Interface is up.
          Broadcast           ; 0x2   ; Broadcast address valid.
          Debug               ; 0x4   ; Turn on debugging.
          Loopback            ; 0x8   ; Is a loopback net.
          Point-to-point      ; 0x10  ; Interface is point-to-point
                                        link.
          No trailers         ; 0x20  ; Avoid use of trailers.
          Resources allocated ; 0x40  ; Resources allocated.
          No ARP              ; 0x80  ; No address resolution protocol.
          Receive all         ; 0x100 ; Receive all packets.
```

7.107. networkInterface

```
   elementId: TBD
   name: networkInterface
   dataType: orderedList
   status: current
   description: Information about a network interface
                installed on an endpoint. The
                following high-level digram
                describes the structure of
                networkInterface information
                element.
   structure: orderedList(interfaceName, interfaceIndex, macAddress,
                          interfaceType, flags)
```

7.108. softwareIdentifier

```
   elementId: TBD
   name: softwareIdentifier
   dataType: string
   status: current
   description: A globally unique identifier for a particular
                software application.
```

7.109. softwareTitle

```
   elementId: TBD
   name: softwareTitle
   dataType: string
   status: current
   description: The title of the software application.
```

7.110. softwareCreator

```
   elementId: TBD
   name: softwareCreator
   dataType: string
   status: current
   description: The software developer (e.g., vendor or author).
```

7.111. simpleSoftwareVersion

```
   elementId: TBD
   name: simpleSoftwareVersion
   dataType: string
   status: current
   description: The version string for a software application that
                conforms to the format of a list of hierarchical
                non-negative integers separated by a single character
                delimiter format.
```

7.112. rpmSoftwareVersion

```
   elementId: TBD
   name: rpmSoftwareVersion
   dataType: string
   status: current
   description: The version string for a software application that
                conforms to the EPOCH:VERSION-RELEASE format.
```

7.113. ciscoTrainSoftwareVersion

```
   elementId: TBD
   name: ciscoTrainSoftwareVersion
   dataType: string
   status: current
   description: The version string for a software application that
                conforms to the Cisco IOS Train string format.
```

7.114. softwareVersion

```
   elementId: TBD
   name: softwareVerison
   dataType: category
   status: current
   description: The version of the software application. Software
                applications may be versioned using a number of
                schemas. The following high-level digram describes
                the structure of the softwareVersion information
                element.
   structure: category(simpleSoftwareVersion | rpmSoftwareVersion |
                   ciscoTrainSoftwareVersion)
```

7.115. softwareLastUpdated

```
   elementId: TBD
   name: softwareLastUpdated
   dataType: dateTimeSeconds
   status: current
   description: The date and time when the software instance
                was last updated on the system (e.g., new
                version instlalled or patch applied)
```

7.116. softwareClass

```
            elementId: TBD
            name: softwareClass
            dataType: enumeration
            status: current
            description: The class of the software instance.
            structure:
        Unknown                ; 0x1 ; The class is not known.
        Other                  ; 0x2 ; The class is known, but,
                                       something other than a value
                                       listed in the enumeration.
        Driver                 ; 0x3 ; The class is a device driver.
        Configuration Software ; 0x4 ; The class is configuration
                                       software.
        Application Software   ; 0x5 ; The class is application
                                       software.
        Instrumentation        ; 0x6 ; The class is instrumentation.
        Diagnostic Software    ; 0x8 ; The class is diagnostic
                                       software.
        Operating System       ; 0x9 ; The class is operating
                                       system.
        Middleware             ; 0xA ; The class is middleware.
        Firmware               ; 0xB ; The class is firmware.
        BIOS/FCode             ; 0xC ; The class is BIOS or FCode.
        Support/Service Pack   ; 0xD ; The class is a support or
                                       service pack.
        Software Bundle        ; 0xE ; The class is a software
                                       bundle.
            References: See Classifications of the DMTF
                        CIM_SoftwareIdentity schema.
```

7.117. softwareInstance

```
elementId: TBD
name: softwareInstance
dataType: orderedList
status: current
description: Information about an instance of software
             installed on an endpoint. The following
             high-level digram describes the structure of
             the softwareInstance information element.
structure: orderedList(softwareIdentifier, softwareTitle,
                       softwareCreator, softwareVersion,
                       softwareLastUpdated, softwareClass)
```

7.118. globallyUniqueIdentifier

elementId: TBD
name: globallyUniqueIdentifier
dataType: unsigned8
status: current
description: TODO.

7.119. creationTimestamp

elementId: TBD
name: creationTimestamp
dataType: dateTimeSeconds
status: current
description: The date and time when the posture
information was created by a SACM Component.

7.120. collectionTimestamp

elementId: TBD
name: collectionTimestamp
dataType: dateTimeSeconds
status: current
description: The date and time when the posture
information was collected or observed by a SACM
Component.

7.121. publicationTimestamp

elementId: TBD
name: publicationTimestamp
dataType: dateTimeSeconds
status: current
description: The date and time when the posture
information was published.

7.122. relayTimestamp

elementId: TBD
name: relayTimestamp
dataType: dateTimeSeconds
status: current
description: The date and time when the posture
information was relayed to another SACM Component.

7.123. storageTimestamp

elementId: TBD
name: storageTimestamp
dataType: dateTimeSeconds
status: current
description: The date and time when the posture
information was stored in a Repository.

7.124. type

elementId: TBD
name: type
dataType: enumeration
status: current
description: The type of data model use to represent
some set of endpoint information. The following
table lists the set of data models supported by SACM.
structure: TBD

7.125. protocolIdentifier

elementId: TBD
name: protocolIdentifier
dataType: unsigned8
status: current
description: The value of the protocol number in the IP packet
header. The protocol number identifies the IP packet
payload type. Protocol numbers are defined in the
IANA Protocol Numbers registry.

In Internet Protocol version 4 (IPv4), this is
carried in the Protocol field. In Internet Protocol
version 6 (IPv6), this is carried in the Next Header
field in the last extension header of the packet.

7.126. sourceTransportPort

elementId: TBD
name: sourceTransportPort
dataType: unsigned16
status: current
description: The source port identifier in the transport header.
For the transport protocols UDP, TCP, and SCTP, this
is the source port number given in the respective
header. This field MAY also be used for future
transport protocols that have 16-bit source port
identifiers.

7.127. sourceIPv4PrefixLength

elementId: TBD
name: sourceIPv4PrefixLength
dataType: unsigned8
status: current
description: The number of contiguous bits that are relevant in the sourceIPv4Prefix Information Element.

7.128. ingressInterface

elementId: TBD
name: ingressInterface
dataType: unsigned32
status: current
description: The index of the IP interface where packets of this Flow are being received. The value matches the value of managed object 'ifIndex' as defined in [RFC2863]. Note that ifIndex values are not assigned statically to an interface and that the interfaces may be renumbered every time the device's management system is re-initialized, as specified in [RFC2863].

7.129. destinationTransportPort

elementId: TBD
name: destinationTransportPort
dataType: unsigned16
status: current
description: The destination port identifier in the transport header. For the transport protocols UDP, TCP, and SCTP, this is the destination port number given in the respective header. This field MAY also be used for future transport protocols that have 16-bit destination port identifiers.

7.130. sourceIPv6PrefixLength

elementId: TBD
name: sourceIPv6PrefixLength
dataType: unsigned8
status: current
description: The number of contiguous bits that are relevant in the sourceIPv6Prefix Information Element.

7.131. sourceIPv4Prefix

elementId: TBD
name: sourceIPv4Prefix
dataType: ipv4Address
status: current
description: IPv4 source address prefix.

7.132. destinationIPv4Prefix

elementId: TBD
name: destinationIPv4Prefix
dataType: ipv4Address
status: current
description: IPv4 destination address prefix.

7.133. sourceMacAddress

elementId: TBD
name: sourceMacAddress
dataType: macAddress
status: current
description: The IEEE 802 source MAC address field.

7.134. ipVersion

elementId: TBD
name: ipVersion
dataType: unsigned8
status: current
description: The IP version field in the IP packet header.

7.135. interfaceDescription

elementId: TBD
name: interfaceDescription
dataType: string
status: current
description: The description of an interface, e.g.
"FastEthernet 1/0" or "ISP connection".

7.136. applicationDescription

elementId: TBD
name: applicationDescription
dataType: string
status: current
description: Specifies the description of an application.

7.137. applicationId

elementId: TBD
name: applicationId
dataType: octetArray
status: current
description: Specifies an Application ID per [RFC6759].

7.138. applicationName

elementId: TBD
name: applicationName
dataType: string
status: current
description: Specifies the name of an application.

7.139. exporterIPv4Address

elementId: TBD
name: exporterIPv4Address
dataType: ipv4Address
status: current
description: The IPv4 address used by the Exporting Process.
This is used by the Collector to identify the
Exporter in cases where the identity of the Exporter
may have been obscured by the use of a proxy.

7.140. exporterIPv6Address

elementId: TBD
name: exporterIPv6Address
dataType: ipv6Address
status: current
description: The IPv6 address used by the Exporting Process.
This is used by the Collector to identify the
Exporter in cases where the identity of the
Exporter may have been obscured by the use of a
proxy.

7.141. portId

```
   elementId: TBD
   name: portId
   dataType: unsigned32
   status: current
   description: An identifier of a line port that is unique per
                IPFIX Device hosting an Observation Point.
                Typically, this Information Element is used for
                limiting the scope of other Information Elements.
```

7.142. templateId

```
   elementId: TBD
   name: templateId
   dataType: unsigned16
   status: current
   description: An identifier of a Template that is locally unique
                within a combination of a Transport session and an
                Observation Domain.

                Template IDs 0-255 are reserved for Template Sets,
                Options Template Sets, and other reserved Sets yet
                to be created. Template IDs of Data Sets are
                numbered from 256 to 65535.

                Typically, this Information Element is used for
                limiting the scope of other Information Elements.
                Note that after a re-start of the Exporting Process
                Template identifiers may be re-assigned.
```

7.143. collectorIPv4Address

```
   elementId: TBD
   name: collectorIPv4Address
   dataType: ipv4Address
   status: current
   description: An IPv4 address to which the Exporting Process sends
                Flow information.
```

7.144. collectorIPv6Address

```
   elementId: TBD
   name: collectorIPv6Address
   dataType: ipv6Address
   status: current
   description: An IPv6 address to which the Exporting Process sends
                Flow information.
```

7.145. informationElementIndex

elementId: TBD
name: informationElementIndex
dataType: unsigned16
status: current
description: A zero-based index of an Information Element
referenced by informationElementId within a Template
referenced by templateId; used to disambiguate
scope for templates containing multiple identical
Information Elements.

7.146. informationElementId

elementId: TBD
name: informationElementId
dataType: unsigned16
status: current
description: This Information Element contains the ID of another
Information Element.

7.147. informationElementDataType

elementId: TBD
name: informationElementDataType
dataType: unsigned8
status: current
description: A description of the abstract data type of an IPFIX
information element.These are taken from the
abstract data types defined in section 3.1 of the
IPFIX Information Model [RFC5102]; see that section
for more information on the types described in the
informationElementDataType sub-registry.

These types are registered in the IANA IPFIX
Information Element Data Type subregistry. This
subregistry is intended to assign numbers for type
names, not to provide a mechanism for adding data
types to the IPFIX Protocol, and as such requires a
Standards Action [RFC5226] to modify.

7.148. informationElementDescription

```
   elementId: TBD
   name: informationElementDescription
   dataType: string
   status: current
   description: A UTF-8 [RFC3629] encoded Unicode string containing
                a human-readable description of an Information
                Element.  The content of the
                informationElementDescription MAY be annotated with
                one or more language tags [RFC4646], encoded
                in-line [RFC2482] within the UTF-8 string, in order
                to specify the language in which the description is
                written.  Description text in multiple languages MAY
                tag each section with its own language tag; in this
                case, the description information in each language
                SHOULD have equivalent meaning.  In the absence of
                any language tag, the "i-default" [RFC2277] language
                SHOULD be assumed.  See the Security Considerations
                section for notes on string handling for Information
                Element type records.
```

7.149. informationElementName

```
   elementId: TBD
   name: informationElementName
   dataType: string
   status: current
   description: A UTF-8 [RFC3629] encoded Unicode string containing
                the name of an Information Element, intended as a
                simple identifier.  See the Security Considerations
                section for notes on string handling for Information
                Element type records.
```

7.150. informationElementRangeBegin

```
   elementId: TBD
   name: informationElementRangeBegin
   dataType: unsigned64
   status: current
   description: Contains the inclusive low end of the range of
                acceptable values for an Information Element.
```

7.151. informationElementRangeEnd

```
   elementId: TBD
   name: informationElementRangeEnd
   dataType: unsigned64
   status: current
   description: Contains the inclusive high end of the range of
                acceptable values for an Information Element.
```

7.152. informationElementSemantics

```
   elementId: TBD
   name: informationElementSemantics
   dataType: unsigned8
   status: current
   description: A description of the semantics of an IPFIX
                Information Element.  These are taken from the data
                type semantics defined in section 3.2 of the IPFIX
                Information Model [RFC5102]; see that section for
                more information on the types defined in the
                informationElementSemantics sub-registry.  This
                field may take the values in Table ; the special
                value 0x00 (default) is used to note that no
                semantics apply to the field; it cannot be
                manipulated by a Collecting Process or File Reader
                that does not understand it a priori.

                These semantics are registered in the IANA IPFIX
                Information Element Semantics subregistry.  This
                subregistry is intended to assign numbers for
                semantics names, not to provide a mechanism for
                adding semantics to the IPFIX Protocol, and as such
                requires a Standards Action [RFC5226] to modify.
```

7.153. informationElementUnits

```
   elementId: TBD
   name: informationElementUnits
   dataType: unsigned16
   status: current
   description: A description of the units of an IPFIX Information
                Element.  These correspond to the units implicitly
                defined in the Information Element definitions in
                section 5 of the IPFIX Information Model [RFC5102];
                see that section for more information on the types
                described in the informationElementsUnits
                sub-registry. This field may take the values in
                Table 3 below; the special value 0x00 (none) is
                used to note that the field is unitless.

                These types are registered in the IANA IPFIX
                Information Element Units subregistry; new types
                may be added on a First Come First Served [RFC5226]
                basis.
```

7.154. applicationCategoryName

```
   elementId: TBD
   name: applicationCategoryName
   dataType: string
   status: current
   description: An attribute that provides a first level
                categorization for each Application ID.
```

7.155. mibObjectValueInteger

```
   elementId: TBD
   name: mibObjectValueInteger
   dataType: signed64
   status: current
   description: An IPFIX Information Element which denotes that the
                integer value of a MIB object will be exported.
                The MIB Object Identifier ("mibObjectIdentifier")
                for this field MUST be exported in a MIB Field
                Option or via another means.  This Information
                Element is used for MIB objects with the Base
                Syntax of Integer32 and INTEGER with IPFIX Reduced
                Size Encoding used as required. The value is
                encoded as per the standard IPFIX Abstract Data Type
                of signed64.
```

7.156. mibObjectValueOctetString

elementId: TBD
name: mibObjectValueOctetString
dataType: octetArray
status: current
description: An IPFIX Information Element which denotes that an Octet String or Opaque value of a MIB object will be exported. The MIB Object Identifier ("mibObjectIdentifier") for this field MUST be exported in a MIB Field Option or via another means. This Information Element is used for MIB objects with the Base Syntax of OCTET STRING and Opaque. The value is encoded as per the standard IPFIX Abstract Data Type of octetArray.

7.157. mibObjectValueOID

elementId: TBD
name: mibObjectValueOID
dataType: octetArray
status: current
description: An IPFIX Information Element which denotes that an Object Identifier or OID value of a MIB object will be exported. The MIB Object Identifier ("mibObjectIdentifier") for this field MUST be exported in a MIB Field Option or via another means. This Information Element is used for MIB objects with the Base Syntax of OBJECT IDENTIFIER. Note - In this case the "mibObjectIdentifier" will define which MIB object is being exported while the value contained in this Information Element will be an OID as a value. The mibObjectValueOID Information Element is encoded as ASN.1/BER [BER] in an octetArray.

7.158. mibObjectValueBits

elementId: TBD
name: mibObjectValueBits
dataType: octetArray
status: current
description: An IPFIX Information Element which denotes that a set of Enumerated flags or bits from a MIB object will be exported. The MIB Object Identifier ("mibObjectIdentifier") for this field MUST be exported in a MIB Field Option or via another means. This Information Element is used for MIB objects with the Base Syntax of BITS. The flags or bits are encoded as per the standard IPFIX Abstract Data Type of octetArray, with sufficient length to accommodate the required number of bits. If the number of bits is not an integer multiple of octets then the most significant bits at end of the octetArray MUST be set to zero.

7.159. mibObjectValueIPAddress

elementId: TBD
name: mibObjectValueIPAddress
dataType: ipv4Address
status: current
description: An IPFIX Information Element which denotes that the IPv4 Address of a MIB object will be exported. The MIB Object Identifier ("mibObjectIdentifier") for this field MUST be exported in a MIB Field Option or via another means. This Information Element is used for MIB objects with the Base Syntax of IPaddress. The value is encoded as per the standard IPFIX Abstract Data Type of ipv4Address.

7.160. mibObjectValueCounter

elementId: TBD
name: mibObjectValueCounter
dataType: unsigned64
status: current
description: An IPFIX Information Element which denotes that the counter value of a MIB object will be exported. The MIB Object Identifier ("mibObjectIdentifier") for this field MUST be exported in a MIB Field Option or via another means. This Information Element is used for MIB objects with the Base Syntax of Counter32 or Counter64 with IPFIX Reduced Size Encoding used as required. The value is encoded as per the standard IPFIX Abstract Data Type of unsigned64.

7.161. mibObjectValueGauge

elementId: TBD
name: mibObjectValueGauge
dataType: unsigned32
status: current
description: An IPFIX Information Element which denotes that the Gauge value of a MIB object will be exported. The MIB Object Identifier ("mibObjectIdentifier") for this field MUST be exported in a MIB Field Option or via another means. This Information Element is used for MIB objects with the Base Syntax of Gauge32. The value is encoded as per the standard IPFIX Abstract Data Type of unsigned64. This value will represent a non-negative integer, which may increase or decrease, but shall never exceed a maximum value, nor fall below a minimum value.

7.162. mibObjectValueTimeTicks

elementId: TBD
name: mibObjectValueTimeTicks
dataType: unsigned32
status: current
description: An IPFIX Information Element which denotes that the TimeTicks value of a MIB object will be exported. The MIB Object Identifier ("mibObjectIdentifier") for this field MUST be exported in a MIB Field Option or via another means. This Information Element is used for MIB objects with the Base Syntax of TimeTicks. The value is encoded as per the standard IPFIX Abstract Data Type of unsigned32.

7.163. mibObjectValueUnsigned

elementId: TBD
name: mibObjectValueUnsigned
dataType: unsigned64
status: current
description: An IPFIX Information Element which denotes that an unsigned integer value of a MIB object will be exported. The MIB Object Identifier ("mibObjectIdentifier") for this field MUST be exported in a MIB Field Option or via another means. This Information Element is used for MIB objects with the Base Syntax of unsigned64 with IPFIX Reduced Size Encoding used as required. The value is encoded as per the standard IPFIX Abstract Data Type of unsigned64.

7.164. mibObjectValueTable

elementId: TBD
name: mibObjectValueTable
dataType: orderedList
status: current
description: An IPFIX Information Element which denotes that a complete or partial conceptual table will be exported. The MIB Object Identifier ("mibObjectIdentifier") for this field MUST be exported in a MIB Field Option or via another means. This Information Element is used for MIB objects with a SYNTAX of SEQUENCE. This is encoded as a subTemplateList of mibObjectValue Information Elements. The template specified in the subTemplateList MUST be an Options Template and MUST include all the Objects listed in the INDEX clause as Scope Fields.
structure: orderedList(mibObjectValueRow+)

7.165. mibObjectValueRow

```
   elementId: TBD
   name: mibObjectValueRow
   dataType: orderedList
   status: current
   description: An IPFIX Information Element which denotes that a
                single row of a conceptual table will be exported.
                The MIB Object Identifier ("mibObjectIdentifier")
                for this field MUST be exported in a MIB Field
                Option or via another means.  This Information
                Element is used for MIB objects with a SYNTAX of
                SEQUENCE.  This is encoded as a subTemplateList of
                mibObjectValue Information Elements.  The
                subTemplateList exported MUST contain exactly one
                row (i.e., one instance of the subtemplate).  The
                template specified in the subTemplateList MUST be
                an Options Template and MUST include all the
                Objects listed in the INDEX clause as Scope Fields.
   structure:   orderedList(mibObjectValue+)
```

7.166. mibObjectIdentifier

```
   elementId: TBD
   name: mibObjectIdentifier
   dataType: octetArray
   status: current
   description: An IPFIX Information Element which denotes that a
                MIB Object Identifier (MIB OID) is exported in the
                (Options) Template Record.  The mibObjectIdentifier
                Information Element contains the OID assigned to
                the MIB Object Type Definition encoded as
                ASN.1/BER [BER].
```

7.167. mibSubIdentifier

```
   elementId: TBD
   name: mibSubIdentifier
   dataType: unsigned32
   status: current
   description: A non-negative sub-identifier of an Object
                Identifier (OID).
```

7.168. mibIndexIndicator

```
   elementId: TBD
   name: mibIndexIndicator
   dataType: unsigned64
   status: current
   description: This set of bit fields is used for marking the
                Information Elements of a Data Record that serve as
                INDEX MIB objects for an indexed Columnar MIB
                object.  Each bit represents an Information Element
                in the Data Record with the n-th bit representing
                the n-th Information Element.  A bit set to value 1
                indicates that the corresponding Information Element
                is an index of the Columnar Object represented by
                the mibFieldValue.  A bit set to value 0 indicates
                that this is not the case.

                If the Data Record contains more than 64
                Information Elements, the corresponding Template
                SHOULD be designed such that all INDEX
                Fields are among the first 64 Information Elements,
                because the mibIndexIndicator only contains 64 bits.
                If the Data Record contains less than 64
                Information Elements, then the extra bits in the
                mibIndexIndicator for which no corresponding
                Information Element exists MUST have the value 0,
                and must be disregarded by the Collector.  This
                Information Element may be exported with
                IPFIX Reduced Size Encoding.
```

7.169. mibCaptureTimeSemantics

elementId: TBD
name: mibCaptureTimeSemantics
dataType: unsigned8
status: current
description: Indicates when in the lifetime of the flow the MIB value was retrieved from the MIB for a mibObjectIdentifier. This is used to indicate if the value exported was collected from the MIB closer to flow creation or flow export time and will refer to the Timestamp fields included in the same record. This field SHOULD be used when exporting a mibObjectValue that specifies counters or statistics.

If the MIB value was sampled by SNMP prior to the IPFIX Metering Process or Exporting Process retrieving the value (i.e., the data is already stale) and it's important to know the exact sampling time, then an additional observationTime* element should be paired with the OID using structured data. Similarly, if different mibCaptureTimeSemantics apply to different mibObject elements within the Data Record, then individual mibCaptureTimeSemantics should be paired with each OID using structured data.

Values:
0. undefined
1. begin - The value for the MIB object is captured from the MIB when the Flow is first observed
2. end - The value for the MIB object is captured from the MIB when the Flow ends
3. export - The value for the MIB object is captured from the MIB at export time
4. average - The value for the MIB object is an average of multiple captures from the MIB over the observed life of the Flow

7.170. mibContextEngineID

elementId: TBD
name: mibContextEngineID
dataType: octetArray
status: current
description: A mibContextEngineID that specifies the SNMP engine ID for a MIB field being exported over IPFIX. Definition as per [RFC3411] section 3.3.

7.171. mibContextName

elementId: TBD
name: mibContextName
dataType: string
status: current
description: This Information Element denotes that a MIB Context Name is specified for a MIB field being exported over IPFIX. Reference [RFC3411] section 3.3.

7.172. mibObjectName

elementId: TBD
name: mibObjectName
dataType: string
status: current
description: The name (called a descriptor in [RFC2578] of an object type definition.

7.173. mibObjectDescription

elementId: TBD
name: mibObjectDescription
dataType: string
status: current
description: The value of the DESCRIPTION clause of an MIB object type definition.

7.174. mibObjectSyntax

elementId: TBD
name: mibObjectSyntax
dataType: string
status: current
description: The value of the SYNTAX clause of an MIB object type definition, which may include a Textual Convention or Subtyping. See [RFC2578].

7.175. mibModuleName

elementId: TBD
name: mibModuleName
dataType: string
status: current
description: The textual name of the MIB module that defines a MIB Object.

7.176. interface

elementId: TBD
name: interface
dataType: list
structure: list (interfaceName, hwAddress, inetAddr, netmask)
status: current
description: Represents an interface and its configuration options.

7.177. iflisteners

elementId: TBD
name: iflisteners
dataType: list
structure: list (interfaceName, physicalProtocol, hwAddress, programName, pid, userId)
status: current
description: Stores the results of checking for applications that are bound to an ethernet interface on the system.

7.178. physicalProtocol

elementId: TBD
name: physicalProtocol
dataType: enumeration
structure:
ETH_P_LOOP ; 0x1 ; Ethernet loopback packet.
ETH_P_PUP ; 0x2 ; Xerox PUP packet.
ETH_P_PUPAT ; 0x3 ; Xerox PUP Address Transport packet.
ETH_P_IP ; 0x4 ; Internet protocol packet.
ETH_P_X25 ; 0x5 ; CCITT X.25 packet.
ETH_P_ARP ; 0x6 ; Address resolution packet.
ETH_P_BPQ ; 0x7 ; G8BPQ AX.25 ethernet packet.
ETH_P_IEEEPUP ; 0x8 ; Xerox IEEE802.3 PUP packet.
ETH_P_IEEEPUPAT ; 0x9 ; Xerox IEEE802.3 PUP address transport packet.
ETH_P_DEC ; 0xA ; DEC assigned protocol.
ETH_P_DNA_DL ; 0xB ; DEC DNA Dump/Load.
ETH_P_DNA_RC ; 0xC ; DEC DNA Remote Console.
ETH_P_DNA_RT ; 0xD ; DEC DNA Routing.
ETH_P_LAT ; 0xE ; DEC LAT.
ETH_P_DIAG ; 0xF ; DEC Diagnostics.
ETH_P_CUST ; 0x10 ; DEC Customer use.
ETH_P_SCA ; 0x11 ; DEC Systems Comms Arch.
ETH_P_RARP ; 0x12 ; Reverse address resolution packet.
ETH_P_ATALK ; 0x13 ; Appletalk DDP.
ETH_P_AARP ; 0x14 ; Appletalk AARP.

```
   ETH_P_8021Q ; 0x15 ; 802.1Q VLAN Extended Header.
   ETH_P_IPX ; 0x16 ; IPX over DIX.
   ETH_P_IPV6 ; 0x17 ; IPv6 over bluebook.
   ETH_P_SLOW ; 0x18 ; Slow Protocol. See 802.3ad 43B.
   ETH_P_WCCP ; 0x19 ; Web-cache coordination protocol.
   ETH_P_PPP_DISC ; 0x1A ; PPPoE discovery messages.
   ETH_P_PPP_SES ; 0x1B ; PPPoE session messages.
   ETH_P_MPLS_UC ; 0x1C ; MPLS Unicast traffic.
   ETH_P_MPLS_MC ; 0x1D ; MPLS Multicast traffic.
   ETH_P_ATMMPOA ; 0x1E ; MultiProtocol Over ATM.
   ETH_P_ATMFATE ; 0x1F ; Frame-based ATM Transport over Ethernet.
   ETH_P_AOE ; 0x20 ; ATA over Ethernet.
   ETH_P_TIPC ; 0x21 ; TIPC.
   ETH_P_802_3 ; 0x22 ; Dummy type for 802.3 frames.
   ETH_P_AX25 ; 0x23 ; Dummy protocol id for AX.25.
   ETH_P_ALL ; 0x24 ; Every packet.
   ETH_P_802_2 ; 0x25 ; 802.2 frames.
   ETH_P_SNAP ; 0x26 ; Internal only.
   ETH_P_DDCMP ; 0x27 ; DEC DDCMP: Internal only
   ETH_P_WAN_PPP ; 0x28 ; Dummy type for WAN PPP frames.
   ETH_P_PPP_MP ; 0x29 ; Dummy type for PPP MP frames.
   ETH_P_PPPTALK ; 0x2A ; Dummy type for Atalk over PPP.
   ETH_P_LOCALTALK ; 0x2B ; Localtalk pseudo type.
   ETH_P_TR_802_2 ; 0x2C ; 802.2 frames.
   ETH_P_MOBITEX ; 0x2D ; Mobitex.
   ETH_P_CONTROL ; 0x2E ; Card specific control frames.
   ETH_P_IRDA ; 0x2F ; Linux-IrDA.
   ETH_P_ECONET ; 0x30 ; Acorn Econet.
   ETH_P_HDLC ; 0x31 ; HDLC frames.
   ETH_P_ARCNET ; 0x32 ; 1A for ArcNet.
                ; 0x33 ; The empty string value is permitted here
                to allow for detailed error reporting.
   status: current
   description: The physical layer protocol used by the AF_PACKET
   socket.
```

7.179. hwAddress

```
   elementId: TBD
   name: hwAddress
   dataType: string
   status: current
   description: The hardware address associated
         with the interface.
```

7.180. programName

elementId: TBD
name: programName
dataType: string
status: current
description: The name of the communicating
program.

7.181. userId

elementId: TBD
name: userId
dataType: unsigned32
status: current
description: The numeric user id.

7.182. inetlisteningserver

elementId: TBD
name: inetlisteningserver
dataType: list
structure: list (transportProtocol, localAddress,
localPort, localFullAddress, programName, foreignAddress,
foreignPort, foreignFullAddress, pid, userId)
status: current
description: Stores the results of checking for network servers
currently active on a system. It holds information pertaining to
a specific protocol-address-port combination.

7.183. transportProtocol

elementId: TBD
name: transportProtocol
dataType: string
status: current
description: The transport-layer
protocol (tcp or udp).

7.184. localAddress

elementId: TBD
name: localAddress
dataType: ipAddress
status: current
description: This is the IP address being listened to. Note that
the IP address can be IPv4 or IPv6.

7.185. localPort

elementId: TBD
name: localPort
dataType: unsigned32
status: current
description: This is the TCP or UDP port
being listened to.

7.186. localFullAddress

elementId: TBD
name: localFullAddress
dataType: string
status: current
description: The IP address and network port on which the program listens, including the local address and the local port. Note that the IP address can be IPv4 or IPv6.

7.187. foreignAddress

elementId: TBD
name: foreignAddress
dataType: ipAddress
status: current
description: The IP address with which the program is communicating, or with which it will communicate. Note that the IP address can be IPv4 or IPv6.

7.188. foreignFullAddress

elementId: TBD
name: foreignFullAddress
dataType: ipAddress
status: current
description: The IP address and network port to which the program is communicating or will accept communications from, including the foreign address and foreign port. Note that the IP address can be IPv4 or IPv6.

7.189. selinuxboolean

```
   elementId: TBD
   name: selinuxboolean
   dataType: list
   structure: list (selinuxName, currentStatus,
         pendingStatus)
   status: current
   description: Describes the current and pending status of a
   SELinux boolean.
```

7.190. selinuxName

```
   elementId: TBD
   name: selinuxName
   dataType: string
   status: current
   description: The name of the SELinux
         boolean.
```

7.191. currentStatus

```
   elementId: TBD
   name: currentStatus
   dataType: boolean
   status: current
   description: Indicates current state of
         the specified SELinux boolean.
```

7.192. pendingStatus

```
   elementId: TBD
   name: pendingStatus
   dataType: boolean
   status: current
   description: Indicates the pending
         state of the specified SELinux boolean.
```

7.193. selinuxsecuritycontext

```
   elementId: TBD
   name: selinuxsecuritycontext
   dataType: list
   structure: list (filepath, path, filename, pid,
         username, role, domainType, lowSensitivity, lowCategory,
         highSensitivity, highCategory, rawlowSensitivity,
         rawlowCategory, rawhighSensitivity, rawhighCategory)
   status: current
   description: Describes the SELinux security
         context of a file or process on the local system.
```

7.194. filepath

elementId: TBD
name: filepath
dataType: string
status: current
description: Specifies the absolute path for a file on the machine. A directory cannot be specified as a filepath.

7.195. path

elementId: TBD
name: path
dataType: string
status: current
description: Specifies the directory component of the absolute path to a file on the machine.

7.196. filename

elementId: TBD
name: filename
dataType: string
status: current
description: The name of the file.

7.197. pid

elementId: TBD
name: pid
dataType: unsigned32
status: current
description: The process ID of the process.

7.198. role

elementId: TBD
name: role
dataType: string
status: current
description: Specifies the types that a process may transition to (domain transitions).

7.199. domainType

elementId: TBD
name: domainType
dataType: string
status: current
description: Specifies the domain in which the file is accessible or the domain in which a process executes.

7.200. lowSensitivity

elementId: TBD
name: lowSensitivity
dataType: string
status: current
description: Specifies the current sensitivity of a file or process.

7.201. lowCategory

elementId: TBD
name: lowCategory
dataType: string
status: current
description: Specifies the set of categories associated with the low sensitivity.

7.202. highSensitivity

elementId: TBD
name: highSensitivity
dataType: string
status: current
description: Specifies the maximum range for a file or the clearance for a process.

7.203. highCategory

elementId: TBD
name: highCategory
dataType: string
status: current
description: Specifies the set of categories associated with the high sensitivity.

7.204. rawlowSensitivity

elementId: TBD
name: rawlowSensitivity
dataType: string
status: current
description: Specifies the current sensitivity of a file or process but in its raw context.

7.205. rawlowCategory

elementId: TBD
name: rawlowCategory
dataType: string
status: current
description: Specifies the set of categories associated with the low sensitivity but in its raw context.

7.206. rawhighSensitivity

elementId: TBD
name: rawhighSensitivity
dataType: string
status: current
description: Specifies the maximum range for a file or the clearance for a process but in its raw context.

7.207. rawhighCategory

elementId: TBD
name: rawhighCategory
dataType: string
status: current
description: Specifies the set of categories associated with the high sensitivity but in its raw context.

7.208. systemdunitdependency

elementId: TBD
name: systemdunitdependency
dataType: list
structure: list (unit, dependency)
status: current

description: Stores the dependencies of the systemd unit.

7.209. unit

elementId: TBD
name: unit
dataType: string
status: current
description: Refers to the full systemd unit name, which has a form of "$name.$type". For example "cupsd.service". This name is usually also the filename of the unit configuration file.

7.210. dependency

elementId: TBD
name: dependency
dataType: string
status: current
description: Refers to the name of a unit that was confirmed to be a dependency of the given unit.

7.211. systemdunitproperty

elementId: TBD
name: systemdunitproperty
dataType: list
structure: list (unit, property, systemdunitValue)

status: current
description: Stores the properties and values of a systemd unit.

7.212. property

elementId: TBD
name: property
dataType: string
status: current
description: The property associated with a
systemd unit.

7.213. systemdunitValue

elementId: TBD
name: systemdunitValue
dataType: string
status: current
description: The value of the property associated with a systemd unit. Exactly one value shall be used for all property types except dbus arrays - each array element shall be represented by one value.

7.214. file

elementId: TBD
name: file
dataType: list
structure: list (filepath, path, filename, fileType, userId, aTime, cTime, mTime, size)
status: current
description: The metadata associated with a file on the endpoint.

7.215. fileType

elementId: TBD
name: fileType
dataType: string
status: current
description: The file's type (e.g., regular file (regular), directory, named pipe (fifo), symbolic link, socket or block special.)

7.216. groupId

elementId: TBD
name: groupId
dataType: unsigned32
status: current
description: The group owner of the file, by group number.

7.217. aTime

elementId: TBD
name: aTime
dataType: dateTimeSeconds
status: current
description: The time that the file was last accessed.

7.218. cTime

elementId: TBD
name: cTime
dataType: dateTimeSeconds
status: current
description: The time of the last change to the file's inode.

7.219. mTime

elementId: TBD
name: mTime
dataType: dateTimeSeconds
status: current
description: The time of the last change to the file's contents.

7.220. size

elementId: TBD
name: size
dataType: unsigned32
status: current
description: This is the size of the file in bytes.

7.221. suid

elementId: TBD
name: suid
dataType: boolean
status: current
description: Indicates whether the program runs with the uid (thus privileges) of the file's owner, rather than the calling user.

7.222. sgid

elementId: TBD
name: sgid
dataType: boolean
status: current
description: Indicates whether the program runs with the gid (thus privileges) of the file's group owner, rather than the calling user's group.

7.223. sticky

elementId: TBD
name: sticky
dataType: boolean
status: current
description: Indicates whether users can delete each other's files in this directory, when said directory is writable by those users.

7.224. hasExtendedAcl

elementId: TBD
name: hasExtendedAcl
dataType: boolean
status: current
description: Indicates whether the file or directory hasACL permissions applied to it. If a system supports ACLs and the file or directory doesn't have an ACL, or it matches the standard UNIX permissions, the entity will have a status of 'exists' and a value of 'false'. If the system supports ACLs and the file or directory has an ACL, the entity will have a status of 'exists' and a value of 'true'. Lastly, if a system doesn't support ACLs, the entity will have a status of 'does not exist'.

7.225. inetd

elementId: TBD
name: inetd
dataType: list
structure: list (serviceProtocol, serviceName, serverProgram, serverArguments, inetdEndpointType, execAsUser, waitStatus)
status: current
description: Holds information associated with different Internet services.

7.226. serverProgram

elementId: TBD
name: serverProgram
dataType: string
status: current
description: Either the pathname of a server program to be invoked by inetd to perform the requested service, or the value internal if inetd itself provides the service.

7.227. inetdEndpointType

```
elementId: TBD
name: inetdEndpointType
dataType: enumeration
structure:
stream ; 0x1 ; The stream value is used to describe a stream
socket.
dgram ; 0x2 ; The dgram value is used to describe a datagram
socket.
raw ; 0x3 ; The raw value is used to describe a raw socket.
seqpacket ; 0x4 ; The seqpacket value is used to describe a
sequenced packet socket.
tli ; 0x5 ; The tli value is used to describe all TLI endpoints.
sunrpc_tcp ; 0x6 ; The sunrpc_tcp value is used to describe all
SUNRPC TCP endpoints.
sunrpc_udp ; 0x7 ; The sunrpc_udp value is used to describe all
SUNRPC UDP endpoints.
 ; 0x8 ; The empty string value is permitted here to allow for
detailed error reporting.
status: current
description: The endpoint type (aka, socket type) associated with
the service.
```

7.228. execAsUser

```
elementId: TBD
name: execAsUser
dataType: string
status: current
description: The user id of the user the
      server program should run under.
```

7.229. waitStatus

elementId: TBD
name: waitStatus
dataType: enumeration
structure: wait ; 0x1 ; The value of 'wait' specifies that the server that is invoked by inetd will take over the listening socket associated with the service, and once launched, inetd will wait for that server to exit, if ever, before it resumes listening for new service requests.

nowait ; 0x2 ; The value of 'nowait' specifies that the server that is invoked by inetd will not wait for any existing server to finish before taking over the listening socket associated with the service.

; 0x3 ; The empty string value is permitted here to allow for detailed error reporting.
status: current
description: Specifies whether the server that is invoked by inetd will take over the listening socket associated with the service, and whether once launched, inetd will wait for that server to exit, if ever, before it resumes listening for new service requests. The legal values are "wait" or "nowait".

7.230. inetAddr

elementId: TBD
name: inetAddr
dataType: ipAddress
status: current
description: The IP address of the specific interface. Note that the IP address can be IPv4 or IPv6.

7.231. netmask

elementId: TBD
name: netmask
dataType: ipAddress
status: current
description: The bitmask used to calculate
the interface's IP network.

7.232. passwordInfo

```
   elementId: TBD
   name: passwordInfo
   dataType: list
   structure: list (username, password, userId, groupId, gcos,
         homeDir, loginShell, lastLogin)
   status: current
   description: Describes user account information for a
         system.
```

7.233. username

```
   elementId: TBD
   name: username
   dataType: string
   status: current
   description: The name of the user.
```

7.234. password

```
   elementId: TBD
   name: password
   dataType: string
   status: current
   description: The encrypted version of the
         user's password.
```

7.235. gcos

```
   elementId: TBD
   name: gcos
   dataType: string
   status: current
   description:
```

7.236. homeDir

```
   elementId: TBD
   name: homeDir
   dataType: string
   status: current
   description: The user's home
         directory.
```

7.237. loginShell

elementId: TBD
name: loginShell
dataType: string
status: current
description: The user’s shell
program.

7.238. lastLogin

elementId: TBD
name: lastLogin
dataType: unsigned32
status: current
description: The date and time when the
last login occurred.

7.239. process

elementId: TBD
name: process
dataType: list
structure: list (commandLine, pid, ppid, priority, startTime)

status: current
description: Information about a process running on an endpoint.

7.240. commandLine

elementId: TBD
name: commandLine
dataType: string
status: current
description: The string used to start the
process. This includes any parameters that are part of the
command line.

7.241. ppid

elementId: TBD
name: ppid
dataType: unsigned32
status: current
description: The process ID of the process’s
parent process.

7.242. priority

elementId: TBD
name: priority
dataType: unsigned32
status: current
description: The scheduling priority with which the process runs.

7.243. startTime

elementId: TBD
name: startTime
dataType: string
status: current
description: The time of day the process started.

7.244. routingtable

elementId: TBD
name: routingtable
dataType: list
structure: list (destination, gateway, flags, interfaceName)
status: current
description: Holds information about an individual routing table entry found in a system’s primary routing table.

7.245. destination

elementId: TBD
name: destination
dataType: ipAddress
status: current
description: The destination IP address prefix of the routing table entry.

7.246. gateway

elementId: TBD
name: gateway
dataType: ipAddress
status: current
description: The gateway of the specified routing table entry.

7.247. runlevelInfo

elementId: TBD
name: runlevelInfo
dataType: list
structure: list (serviceName, runlevel, start, kill)

status: current
description: Information about the start or kill state of a specified service at a given runlevel.

7.248. runlevel

elementId: TBD
name: runlevel
dataType: string
status: current
description: Specifies the system runlevel associated with a service.

7.249. start

elementId: TBD
name: start
dataType: boolean
status: current
description: Specifies whether the service is scheduled to start at the runlevel.

7.250. kill

elementId: TBD
name: kill
dataType: boolean
status: current
description: Specifies whether the service is scheduled to be killed at the runlevel.

7.251. shadowItem

elementId: TBD
name: shadowItem
dataType: list
structure: list (username, password, chgLst, chgAllow, chgReq, expWarn, expInact, expDate, flags, encryptMethod)
status: current
description:

7.252. chgLst

elementId: TBD
name: chgLst
dataType: dateTimeSeconds
status: current
description: The date of the last password
change.

7.253. chgAllow

elementId: TBD
name: chgAllow
dataType: unsigned32
status: current
description: Specifies how often in days a
user may change their password. It can also be thought of
as the minimum age of a password.

7.254. chgReq

elementId: TBD
name: chgReq
dataType: unsigned32
status: current
description: Describes how long a user can
keep a password before the system forces her to change it.

7.255. expWarn

elementId: TBD
name: expWarn
dataType: unsigned32
status: current
description: Describes how long before
password expiration the system begins warning the user.

7.256. expInact

elementId: TBD
name: expInact
dataType: unsigned32
status: current
description: Describes how many days of
account inactivity the system will wait after a password
expires before locking the account.

7.257. expDate

elementId: TBD
name: expDate
dataType: dateTimeSeconds
status: current
description: Specifies when will the
account's password expire.

7.258. encryptMethod

elementId: TBD
name: encryptMethod
dataType: enumeration
structure: DES ; 0x1 ; The DES method corresponds to the (none)
prefix.
BSDi ; 0x2 ; The BSDi method corresponds to BSDi modified
DES or the '_' prefix.
MD5 ; 0x3 ; The MD5 method corresponds to MD5 for Linux/BSD
or the 1 prefix.
Blowfish ; 0x4 ; The Blowfish method corresponds to Blowfish
(OpenBSD) or the 2 or $2a$ prefixes.
Sun MD5 ; 0x5 ; The Sun MD5 method corresponds to the $md5$
prefix.
SHA-256 ; 0x6 ; The SHA-256 method corresponds to the 5
prefix.
SHA-512 ; 0x7 ; The SHA-512 method corresponds to the 6
prefix. ; 0x8 ; The empty string value is permitted here to
allow for empty elements associated with variable references.
status: current
description: Describes method that is used for hashing
passwords.

7.259. symlink

elementId: TBD
name: symlink
dataType: list
structure: list (symlinkFilepath, canonicalPath)
status: current

description: Identifies the result generated for a symlink.

7.260. symlinkFilepath

elementId: TBD
name: symlinkFilepath
dataType: string
status: current
description: Specifies the filepath to
the subject symbolic link file.

7.261. canonicalPath

elementId: TBD
name: canonicalPath
dataType: string
status: current
description: Specifies the canonical
path for the target of the symbolic link file specified by
the filepath.

7.262. sysctl

elementId: TBD
name: sysctl
dataType: list
structure: list (kernelParameterName, kernelParameterValue+,
uname, machineClass, nodeName, osName, osRelease,
osVersion, processorType)
status: current
description: Stores
information retrieved from the local system about a kernel
parameter and its respective value(s).

7.263. kernelParameterName

elementId: TBD
name: kernelParameterName
dataType: string
status: current
description: The name of a kernel
parameter that was collected from the local system.

7.264. kernelParameterValue

elementId: TBD
name: kernelParameterValue
dataType: string
status: current
description: The current value(s)
for the specified kernel parameter on the local system.

7.265. uname

elementId: TBD
name: uname
dataType: list
structure: list (machineClass, nodeName, osName, osRelease, osVersion, processorType)
status: current
description: Information about the hardware the machine is running on.

7.266. machineClass

elementId: TBD
name: machineClass
dataType: string
status: current
description: Specifies the machine hardware name.

7.267. nodeName

elementId: TBD
name: nodeName
dataType: string
status: current
description: Specifies the host name.

7.268. osName

elementId: TBD
name: osName
dataType: string
status: current
description: Specifies the operating system name.

7.269. osRelease

elementId: TBD
name: osRelease
dataType: string
status: current
description: Specifies the build version.

7.270. processorType

elementId: TBD
name: processorType
dataType: string
status: current
description: Specifies the processor
type.

7.271. internetService

elementId: TBD
name: internetService
dataType: list
structure: list (serviceProtocol, serviceName, flags,
noAccess, onlyFrom, port, server, serverArguments,
socketType, registeredServiceType, user, wait, disabled)

status: current
description: Holds information associated with Internet services.

7.272. serviceProtocol

elementId: TBD
name: serviceProtocol
dataType: string
status: current
description: Specifies the protocol
that is used by the service.

7.273. serviceName

elementId: TBD
name: serviceName
dataType: string
status: current
description: Specifies the name of the
service.

7.274. flags

elementId: TBD
name: flags
dataType: string
status: current
description: Specifies miscellaneous settings
associated with the service with executing a program.

7.275. noAccess

elementId: TBD
name: noAccess
dataType: string
status: current
description: Specifies the remote hosts to
which the service is unavailable.

7.276. onlyFrom

elementId: TBD
name: onlyFrom
dataType: ipAddress
status: current
description: Specifies the remote hosts to
which the service is available.

7.277. port

elementId: TBD
name: port
dataType: unsigned32
status: current
description: The port entity specifies the port
used by the service.

7.278. server

elementId: TBD
name: server
dataType: string
status: current
description: Specifies the executable that is
used to launch the service.

7.279. serverArguments

elementId: TBD
name: serverArguments
dataType: string
status: current
description: Specifies the arguments
that are passed to the executable when launching the service.

7.280. socketType

elementId: TBD
name: socketType
dataType: string
status: current
description: Specifies the type of socket
that is used by the service. Possible values include: stream, dgram, raw, or seqpacket.

7.281. registeredServiceType

elementId: TBD
name: registeredServiceType
dataType: enumeration
structure: INTERNAL ; 0x1 ; The INTERNAL type is used to describe services like echo, chargen, and others whose functionality is supplied by xinetd itself.
RPC ; 0x2 ; The RPC type is used to describe services that use remote procedure call ala NFS.
UNLISTED ; 0x3 ; The UNLISTED type is used to describe services that aren't listed in /etc/protocols or /etc/rpc.
TCPMUX ; 0x4 ; The TCPMUX type is used to describe services that conform to RFC 1078. This type indiciates that the service is responsible for handling the protocol handshake.
TCPMUXPLUS ; 0x5 ; The TCPMUXPLUS type is used to describe services that conform to RFC 1078. This type indicates that xinetd is responsible for handling the protocol handshake.
; 0x6 ; The empty string value is permitted here to allow for detailed error reporting.
status: current

description: Specifies the type of internet service.

7.282. wait

elementId: TBD
name: wait
dataType: boolean
status: current
description: Specifies whether or not the service is single-threaded or multi-threaded and whether or not xinetd accepts the connection or the service accepts the connection. A value of 'true' indicates that the service is single-threaded and the service will accept the connection. A value of 'false' indicates that the service is multi-threaded and xinetd will accept the connection.

7.283. disabled

elementId: TBD
name: disabled
dataType: boolean
status: current
description: Specifies whether or not the service is disabled. A value of 'true' indicates that the service is disabled and will not start. A value of 'false' indicates that the service is not disabled.

7.284. windowsView

elementId: TBD
name: windowsView
dataType: enumeration
structure: 32_bit ; 0x1 ; Indicates the 32_bit windows view.
64_bit ; 0x2 ; Indicates the 64_bit windows view.
; 0x3 ; The empty string value is permitted here to allow for empty elements associated with error conditions.
status: current
description: Indicates from which view (32-bit or 64-bit), the information was collected. A value of '32_bit' indicates the Item was collected from the 32-bit view. A value of '64-bit' indicates the Item was collected from the 64-bit view.

7.285. fileauditedpermissions

elementId: TBD
name: fileauditedpermissions
dataType: list
structure: list (filepath, path, filename, trusteeSid, trusteeName, auditStandardDelete, auditStandardReadControl, auditStandardWriteDac, auditStandardWriteOwner, auditStandardSynchronize, auditAccessSystemSecurity, auditGenericRead, auditGenericWrite, auditGenericExecute, auditGenericAll, auditFileReadData, auditFileWriteData, auditFileAppendData, auditFileReadEa, auditFileWriteEa, auditFileExecute, auditFileDeleteChild, auditFileReadAttributes, auditFileWriteAttributes, windowsView)
status: current
description: Stores the audited access rights of a file that a system access control list (SACL) structure grants to a specified trustee. The trustee's audited access rights are determined checking all access control entries (ACEs) in the SACL.

7.286. trusteeName

elementId: TBD
name: trusteeName
dataType: string
status: current
description: Specifies the trustee name. A
trustee can be a user, group, or program (such as a Windows
service).

7.287. auditStandardDelete

elementId: TBD
name: auditStandardDelete
dataType: enumeration
structure: AUDIT_FAILURE ; 0x1 ; The audit type AUDIT_FAILURE is
used to perform audits on all unsuccessful occurrences of
specified events when auditing is enabled.
AUDIT_NONE ; 0x2 ; The audit type AUDIT_NONE is used to cancel
all auditing options for the specified events.
AUDIT_SUCCESS ; 0x3 ; The audit type AUDIT_SUCCESS is used to
perform audits on all successful occurrences of the specified
events when auditing is enabled.
AUDIT_SUCCESS_FAILURE ; 0x4 ; The audit type AUDIT_SUCCESS_FAILURE
is used to perform audits on all successful and unsuccessful
occurrences of the specified events when auditing is enabled.
; 0x5 ; The empty string value is permitted here to allow for
detailed error reporting.
status: current
description: The right to delete the object.

7.288. auditStandardReadControl

elementId: TBD
name: auditStandardReadControl
dataType: enumeration
structure: AUDIT_FAILURE ; 0x1 ; The audit type AUDIT_FAILURE is used to perform audits on all unsuccessful occurrences of specified events when auditing is enabled.
AUDIT_NONE ; 0x2 ; The audit type AUDIT_NONE is used to cancel all auditing options for the specified events.
AUDIT_SUCCESS ; 0x3 ; The audit type AUDIT_SUCCESS is used to perform audits on all successful occurrences of the specified events when auditing is enabled.
AUDIT_SUCCESS_FAILURE ; 0x4 ; The audit type AUDIT_SUCCESS_FAILURE is used to perform audits on all successful and unsuccessful occurrences of the specified events when auditing is enabled.
; 0x5 ; The empty string value is permitted here to allow for detailed error reporting.
status: current
description: The right to read the information in the object’s security descriptor, not including the information in the SACL.

7.289. auditStandardWriteDac

elementId: TBD
name: auditStandardWriteDac
dataType: enumeration
structure: AUDIT_FAILURE ; 0x1 ; The audit type AUDIT_FAILURE is used to perform audits on all unsuccessful occurrences of specified events when auditing is enabled.
AUDIT_NONE ; 0x2 ; The audit type AUDIT_NONE is used to cancel all auditing options for the specified events.
AUDIT_SUCCESS ; 0x3 ; The audit type AUDIT_SUCCESS is used to perform audits on all successful occurrences of the specified events when auditing is enabled.
AUDIT_SUCCESS_FAILURE ; 0x4 ; The audit type AUDIT_SUCCESS_FAILURE is used to perform audits on all successful and unsuccessful occurrences of the specified events when auditing is enabled.
; 0x5 ; The empty string value is permitted here to allow for detailed error reporting.
status: current
description: The right to modify the DACL in the object’s security descriptor.

7.290. auditStandardWriteOwner

elementId: TBD
name: auditStandardWriteOwner
dataType: enumeration
structure: AUDIT_FAILURE ; 0x1 ; The audit type AUDIT_FAILURE is used to perform audits on all unsuccessful occurrences of specified events when auditing is enabled.
AUDIT_NONE ; 0x2 ; The audit type AUDIT_NONE is used to cancel all auditing options for the specified events.
AUDIT_SUCCESS ; 0x3 ; The audit type AUDIT_SUCCESS is used to perform audits on all successful occurrences of the specified events when auditing is enabled.
AUDIT_SUCCESS_FAILURE ; 0x4 ; The audit type AUDIT_SUCCESS_FAILURE is used to perform audits on all successful and unsuccessful occurrences of the specified events when auditing is enabled.
; 0x5 ; The empty string value is permitted here to allow for detailed error reporting.
status: current
description: The right to change the owner in the object’s security descriptor.

7.291. auditStandardSynchronize

elementId: TBD
name: auditStandardSynchronize
dataType: enumeration
structure: AUDIT_FAILURE ; 0x1 ; The audit type AUDIT_FAILURE is used to perform audits on all unsuccessful occurrences of specified events when auditing is enabled.
AUDIT_NONE ; 0x2 ; The audit type AUDIT_NONE is used to cancel all auditing options for the specified events.
AUDIT_SUCCESS ; 0x3 ; The audit type AUDIT_SUCCESS is used to perform audits on all successful occurrences of the specified events when auditing is enabled.
AUDIT_SUCCESS_FAILURE ; 0x4 ; The audit type AUDIT_SUCCESS_FAILURE is used to perform audits on all successful and unsuccessful occurrences of the specified events when auditing is enabled.
; 0x5 ; The empty string value is permitted here to allow for detailed error reporting.
status: current
description: The right to use the object for synchronization. This enables a thread to wait until the object is in the signaled state. Some object types do not support this access right.

7.292. auditAccessSystemSecurity

elementId: TBD
name: auditAccessSystemSecurity
dataType: enumeration
structure: AUDIT_FAILURE ; 0x1 ; The audit type AUDIT_FAILURE is used to perform audits on all unsuccessful occurrences of specified events when auditing is enabled.
AUDIT_NONE ; 0x2 ; The audit type AUDIT_NONE is used to cancel all auditing options for the specified events.
AUDIT_SUCCESS ; 0x3 ; The audit type AUDIT_SUCCESS is used to perform audits on all successful occurrences of the specified events when auditing is enabled.
AUDIT_SUCCESS_FAILURE ; 0x4 ; The audit type AUDIT_SUCCESS_FAILURE is used to perform audits on all successful and unsuccessful occurrences of the specified events when auditing is enabled.
; 0x5 ; The empty string value is permitted here to allow for detailed error reporting.
status: current
description: Indicates access to a system access control list (SACL).

7.293. auditGenericRead

elementId: TBD
name: auditGenericRead
dataType: enumeration
structure: AUDIT_FAILURE ; 0x1 ; The audit type AUDIT_FAILURE is used to perform audits on all unsuccessful occurrences of specified events when auditing is enabled.
AUDIT_NONE ; 0x2 ; The audit type AUDIT_NONE is used to cancel all auditing options for the specified events.
AUDIT_SUCCESS ; 0x3 ; The audit type AUDIT_SUCCESS is used to perform audits on all successful occurrences of the specified events when auditing is enabled.
AUDIT_SUCCESS_FAILURE ; 0x4 ; The audit type AUDIT_SUCCESS_FAILURE is used to perform audits on all successful and unsuccessful occurrences of the specified events when auditing is enabled.
; 0x5 ; The empty string value is permitted here to allow for detailed error reporting.
status: current
description: Read access.

7.294. auditGenericWrite

elementId: TBD
name: auditGenericWrite
dataType: enumeration
structure: AUDIT_FAILURE ; 0x1 ; The audit type AUDIT_FAILURE is used to perform audits on all unsuccessful occurrences of specified events when auditing is enabled.
AUDIT_NONE ; 0x2 ; The audit type AUDIT_NONE is used to cancel all auditing options for the specified events.
AUDIT_SUCCESS ; 0x3 ; The audit type AUDIT_SUCCESS is used to perform audits on all successful occurrences of the specified events when auditing is enabled.
AUDIT_SUCCESS_FAILURE ; 0x4 ; The audit type AUDIT_SUCCESS_FAILURE is used to perform audits on all successful and unsuccessful occurrences of the specified events when auditing is enabled.
; 0x5 ; The empty string value is permitted here to allow for detailed error reporting.
status: current
description: Write access.

7.295. auditGenericExecute

elementId: TBD
name: auditGenericExecute
dataType: enumeration
structure: AUDIT_FAILURE ; 0x1 ; The audit type AUDIT_FAILURE is used to perform audits on all unsuccessful occurrences of specified events when auditing is enabled.
AUDIT_NONE ; 0x2 ; The audit type AUDIT_NONE is used to cancel all auditing options for the specified events.
AUDIT_SUCCESS ; 0x3 ; The audit type AUDIT_SUCCESS is used to perform audits on all successful occurrences of the specified events when auditing is enabled.
AUDIT_SUCCESS_FAILURE ; 0x4 ; The audit type AUDIT_SUCCESS_FAILURE is used to perform audits on all successful and unsuccessful occurrences of the specified events when auditing is enabled.
; 0x5 ; The empty string value is permitted here to allow for detailed error reporting.
status: current
description: Execute access.

7.296. auditGenericAll

elementId: TBD
name: auditGenericAll
dataType: enumeration
structure: AUDIT_FAILURE ; 0x1 ; The audit type AUDIT_FAILURE is used to perform audits on all unsuccessful occurrences of specified events when auditing is enabled.
AUDIT_NONE ; 0x2 ; The audit type AUDIT_NONE is used to cancel all auditing options for the specified events.
AUDIT_SUCCESS ; 0x3 ; The audit type AUDIT_SUCCESS is used to perform audits on all successful occurrences of the specified events when auditing is enabled.
AUDIT_SUCCESS_FAILURE ; 0x4 ; The audit type AUDIT_SUCCESS_FAILURE is used to perform audits on all successful and unsuccessful occurrences of the specified events when auditing is enabled.
; 0x5 ; The empty string value is permitted here to allow for detailed error reporting.
status: current
description: Read, write, and execute access.

7.297. auditFileReadData

elementId: TBD
name: auditFileReadData
dataType: enumeration
structure: AUDIT_FAILURE ; 0x1 ; The audit type AUDIT_FAILURE is used to perform audits on all unsuccessful occurrences of specified events when auditing is enabled.
AUDIT_NONE ; 0x2 ; The audit type AUDIT_NONE is used to cancel all auditing options for the specified events.
AUDIT_SUCCESS ; 0x3 ; The audit type AUDIT_SUCCESS is used to perform audits on all successful occurrences of the specified events when auditing is enabled.
AUDIT_SUCCESS_FAILURE ; 0x4 ; The audit type AUDIT_SUCCESS_FAILURE is used to perform audits on all successful and unsuccessful occurrences of the specified events when auditing is enabled.
; 0x5 ; The empty string value is permitted here to allow for detailed error reporting.
status: current
description: Grants the right to read data from the file.

7.298. auditFileWriteData

elementId: TBD
name: auditFileWriteData
dataType: enumeration
structure: AUDIT_FAILURE ; 0x1 ; The audit type AUDIT_FAILURE is used to perform audits on all unsuccessful occurrences of specified events when auditing is enabled.
AUDIT_NONE ; 0x2 ; The audit type AUDIT_NONE is used to cancel all auditing options for the specified events.
AUDIT_SUCCESS ; 0x3 ; The audit type AUDIT_SUCCESS is used to perform audits on all successful occurrences of the specified events when auditing is enabled.
AUDIT_SUCCESS_FAILURE ; 0x4 ; The audit type AUDIT_SUCCESS_FAILURE is used to perform audits on all successful and unsuccessful occurrences of the specified events when auditing is enabled.
; 0x5 ; The empty string value is permitted here to allow for detailed error reporting.
status: current
description: Grants the right to write data to the file.

7.299. auditFileAppendData

elementId: TBD
name: auditFileAppendData
dataType: enumeration
structure: AUDIT_FAILURE ; 0x1 ; The audit type AUDIT_FAILURE is used to perform audits on all unsuccessful occurrences of specified events when auditing is enabled.
AUDIT_NONE ; 0x2 ; The audit type AUDIT_NONE is used to cancel all auditing options for the specified events.
AUDIT_SUCCESS ; 0x3 ; The audit type AUDIT_SUCCESS is used to perform audits on all successful occurrences of the specified events when auditing is enabled.
AUDIT_SUCCESS_FAILURE ; 0x4 ; The audit type AUDIT_SUCCESS_FAILURE is used to perform audits on all successful and unsuccessful occurrences of the specified events when auditing is enabled.
; 0x5 ; The empty string value is permitted here to allow for detailed error reporting.
status: current
description: Grants the right to append data to the file.

7.300. auditFileReadEa

elementId: TBD
name: auditFileReadEa
dataType: enumeration
structure: AUDIT_FAILURE ; 0x1 ; The audit type AUDIT_FAILURE is used to perform audits on all unsuccessful occurrences of specified events when auditing is enabled.
AUDIT_NONE ; 0x2 ; The audit type AUDIT_NONE is used to cancel all auditing options for the specified events.
AUDIT_SUCCESS ; 0x3 ; The audit type AUDIT_SUCCESS is used to perform audits on all successful occurrences of the specified events when auditing is enabled.
AUDIT_SUCCESS_FAILURE ; 0x4 ; The audit type AUDIT_SUCCESS_FAILURE is used to perform audits on all successful and unsuccessful occurrences of the specified events when auditing is enabled.
; 0x5 ; The empty string value is permitted here to allow for detailed error reporting.
status: current
description: Grants the right to read extended attributes.

7.301. auditFileWriteEa

elementId: TBD
name: auditFileWriteEa
dataType: enumeration
structure: AUDIT_FAILURE ; 0x1 ; The audit type AUDIT_FAILURE is used to perform audits on all unsuccessful occurrences of specified events when auditing is enabled.
AUDIT_NONE ; 0x2 ; The audit type AUDIT_NONE is used to cancel all auditing options for the specified events.
AUDIT_SUCCESS ; 0x3 ; The audit type AUDIT_SUCCESS is used to perform audits on all successful occurrences of the specified events when auditing is enabled.
AUDIT_SUCCESS_FAILURE ; 0x4 ; The audit type AUDIT_SUCCESS_FAILURE is used to perform audits on all successful and unsuccessful occurrences of the specified events when auditing is enabled.
; 0x5 ; The empty string value is permitted here to allow for detailed error reporting.
status: current
description: Grants the right to write extended attributes.

7.302. auditFileExecute

elementId: TBD
name: auditFileExecute
dataType: enumeration
structure: AUDIT_FAILURE ; 0x1 ; The audit type AUDIT_FAILURE is used to perform audits on all unsuccessful occurrences of specified events when auditing is enabled.
AUDIT_NONE ; 0x2 ; The audit type AUDIT_NONE is used to cancel all auditing options for the specified events.
AUDIT_SUCCESS ; 0x3 ; The audit type AUDIT_SUCCESS is used to perform audits on all successful occurrences of the specified events when auditing is enabled.
AUDIT_SUCCESS_FAILURE ; 0x4 ; The audit type AUDIT_SUCCESS_FAILURE is used to perform audits on all successful and unsuccessful occurrences of the specified events when auditing is enabled.
; 0x5 ; The empty string value is permitted here to allow for detailed error reporting.
status: current
description: Grants the right to execute a file.

7.303. auditFileDeleteChild

elementId: TBD
name: auditFileDeleteChild
dataType: enumeration
structure: AUDIT_FAILURE ; 0x1 ; The audit type AUDIT_FAILURE is used to perform audits on all unsuccessful occurrences of specified events when auditing is enabled.
AUDIT_NONE ; 0x2 ; The audit type AUDIT_NONE is used to cancel all auditing options for the specified events.
AUDIT_SUCCESS ; 0x3 ; The audit type AUDIT_SUCCESS is used to perform audits on all successful occurrences of the specified events when auditing is enabled.
AUDIT_SUCCESS_FAILURE ; 0x4 ; The audit type AUDIT_SUCCESS_FAILURE is used to perform audits on all successful and unsuccessful occurrences of the specified events when auditing is enabled.
; 0x5 ; The empty string value is permitted here to allow for detailed error reporting.
status: current
description: Right to delete a directory and all the files it contains (its children), even if the files are read-only.

7.304. auditFileReadAttributes

elementId: TBD
name: auditFileReadAttributes
dataType: enumeration
structure: AUDIT_FAILURE ; 0x1 ; The audit type AUDIT_FAILURE is used to perform audits on all unsuccessful occurrences of specified events when auditing is enabled.
AUDIT_NONE ; 0x2 ; The audit type AUDIT_NONE is used to cancel all auditing options for the specified events.
AUDIT_SUCCESS ; 0x3 ; The audit type AUDIT_SUCCESS is used to perform audits on all successful occurrences of the specified events when auditing is enabled.
AUDIT_SUCCESS_FAILURE ; 0x4 ; The audit type AUDIT_SUCCESS_FAILURE is used to perform audits on all successful and unsuccessful occurrences of the specified events when auditing is enabled.
; 0x5 ; The empty string value is permitted here to allow for detailed error reporting.
status: current
description: Grants the right to read file attributes.

7.305. auditFileWriteAttributes

elementId: TBD
name: auditFileWriteAttributes
dataType: enumeration
structure: AUDIT_FAILURE ; 0x1 ; The audit type AUDIT_FAILURE is used to perform audits on all unsuccessful occurrences of specified events when auditing is enabled.
AUDIT_NONE ; 0x2 ; The audit type AUDIT_NONE is used to cancel all auditing options for the specified events.
AUDIT_SUCCESS ; 0x3 ; The audit type AUDIT_SUCCESS is used to perform audits on all successful occurrences of the specified events when auditing is enabled.
AUDIT_SUCCESS_FAILURE ; 0x4 ; The audit type AUDIT_SUCCESS_FAILURE is used to perform audits on all successful and unsuccessful occurrences of the specified events when auditing is enabled.
; 0x5 ; The empty string value is permitted here to allow for detailed error reporting.
status: current
description: Grants the right to change file attributes.

7.306. fileeffectiverights

```
   elementId: TBD
   name: fileeffectiverights
   dataType: list
   structure: list (filepath, path, filename,
         trusteeSid, trusteeName, standardDelete, standardReadControl,
         standardWriteDac, standardWriteOwner,
         standardSynchronize, accessSystemSecurity, genericRead,
         genericWrite, genericExecute, genericAll, fileReadData,
         fileWriteData, fileAppendData, fileReadEa, fileWriteEa,
         fileExecute, fileDeleteChild, fileReadAttributes,
         fileWriteAttributes, windowsView)
   status: current
   description: Stores the effective rights of a file that a
         discretionary access control list (DACL) structure grants
         to a specified trustee. The trustee's effective rights
         are determined checking all access-allowed and access-denied
         access control entries (ACEs) in the DACL.
```

7.307. standardDelete

```
   elementId: TBD
   name: standardDelete
   dataType: boolean
   status: current
   description: The right to delete the
         object.
```

7.308. standardReadControl

```
   elementId: TBD
   name: standardReadControl
   dataType: boolean
   status: current
   description: The right to read
         the information in the object's security descriptor, not
         including the information in the SACL.
```

7.309. standardWriteDac

```
   elementId: TBD
   name: standardWriteDac
   dataType: boolean
   status: current
   description: The right to modify the
         DACL in the object's security descriptor.
```

7.310. standardWriteOwner

elementId: TBD
name: standardWriteOwner
dataType: boolean
status: current
description: The right to change
the owner in the object's security descriptor.

7.311. standardSynchronize

elementId: TBD
name: standardSynchronize
dataType: boolean
status: current
description: The right to use the
object for synchronization. This enables a thread to wait
until the object is in the signaled state. Some object
types do not support this access right.

7.312. accessSystemSecurity

elementId: TBD
name: accessSystemSecurity
dataType: boolean
status: current
description: Indicates access to
a system access control list (SACL).

7.313. genericRead

elementId: TBD
name: genericRead
dataType: boolean
status: current
description: Read access.

7.314. genericWrite

elementId: TBD
name: genericWrite
dataType: boolean
status: current
description: Write access.

7.315. genericExecute

elementId: TBD
name: genericExecute
dataType: boolean
status: current
description: Execute access.

7.316. genericAll

elementId: TBD
name: genericAll
dataType: boolean
status: current
description: Read, write, and execute
access.

7.317. fileReadData

elementId: TBD
name: fileReadData
dataType: boolean
status: current
description: Grants the right to read
data from the file

7.318. fileWriteData

elementId: TBD
name: fileWriteData
dataType: boolean
status: current
description: Grants the right to write
data to the file.

7.319. fileAppendData

elementId: TBD
name: fileAppendData
dataType: boolean
status: current
description: Grants the right to
append data to the file.

7.320. fileReadEa

elementId: TBD
name: fileReadEa
dataType: boolean
status: current
description: Grants the right to read
extended attributes.

7.321. fileWriteEa

elementId: TBD
name: fileWriteEa
dataType: boolean
status: current
description: Grants the right to write
extended attributes.

7.322. fileExecute

elementId: TBD
name: fileExecute
dataType: boolean
status: current
description: Grants the right to execute
a file.

7.323. fileDeleteChild

elementId: TBD
name: fileDeleteChild
dataType: boolean
status: current
description: Right to delete a
directory and all the files it contains (its children),
even if the files are read-only.

7.324. fileReadAttributes

elementId: TBD
name: fileReadAttributes
dataType: boolean
status: current
description: Grants the right to
read file attributes.

7.325. fileWriteAttributes

elementId: TBD
name: fileWriteAttributes
dataType: boolean
status: current
description: Grants the right to
change file attributes.

7.326. groupInfo

elementId: TBD
name: groupInfo
dataType: list
structure: list (group, username, subgroup)
status: current
description: Specifies the different users and subgroups, that
directly belong to specific groups.

7.327. group

elementId: TBD
name: group
dataType: string
status: current
description: Represents the name of a particular
group.

7.328. subgroup

elementId: TBD
name: subgroup
dataType: string
status: current
description: Represents the name of a
particular subgroup in the specified group.

7.329. groupSidInfo

elementId: TBD
name: groupSidInfo
dataType: list
structure: list (groupSid, userSid, subgroupSid)
status: current
description: Specifies the different users and subgroups, that
directly belong to specific groups
(identified by SID).

7.330. userSidInfo

elementId: TBD
name: userSidInfo
dataType: list
structure: list (userSid, enabled, groupSid, lastLogon)

status: current
description: Specifies the different groups (identified by SID) that a user belongs to.

7.331. userSid

elementId: TBD
name: userSid
dataType: string
status: current
description: Represents the SID of a particular user.

7.332. subgroupSid

elementId: TBD
name: subgroupSid
dataType: string
status: current
description: Represents the SID of a particular subgroup.

7.333. lockoutpolicy

elementId: TBD
name: lockoutpolicy
dataType: list
structure: list (forceLogoff, lockoutDuration, lockoutObservationWindow, lockoutThreshold)
status: current
description: Specifies various attributes associated with lockout information for users and global groups in the security database.

7.334. forceLogoff

elementId: TBD
name: forceLogoff
dataType: unsigned32
status: current
description: Specifies, in seconds, the
amount of time between the end of the valid logon time and
the time when the user is forced to log off the
network.

7.335. lockoutDuration

elementId: TBD
name: lockoutDuration
dataType: unsigned32
status: current
description: Specifies, in seconds,
how long a locked account remains locked before it is
automatically unlocked.

7.336. lockoutObservationWindow

elementId: TBD
name: lockoutObservationWindow
dataType: unsigned32
status: current
description: Specifies the
maximum time, in seconds, that can elapse between any two
failed logon attempts before lockout occurs.

7.337. lockoutThreshold

elementId: TBD
name: lockoutThreshold
dataType: unsigned32
status: current
description: Specifies the number of
invalid password authentications that can occur before an
account is marked "locked out."

7.338. passwordpolicy

```
   elementId: TBD
   name: passwordpolicy
   dataType: list
   structure: list (maxPasswdAge, minPasswdAge,
         minPasswdLen, passwordHistLen, passwordComplexity,
         reversibleEncryption)
   status: current
   description: Specifies
         policy information associated with passwords.
```

7.339. maxPasswdAge

```
   elementId: TBD
   name: maxPasswdAge
   dataType: unsigned32
   status: current
   description: Specifies, in seconds (from
         a DWORD), the maximum allowable password age. A value of
         TIMEQ_FOREVER (max DWORD value, 4294967295) indicates
         that the password never expires. The minimum valid value
         for this element is ONE_DAY (86400). See the
         USER_MODALS_INFO_0 structure returned by a call to
         NetUserModalsGet().
```

7.340. minPasswdAge

```
   elementId: TBD
   name: minPasswdAge
   dataType: unsigned32
   status: current
   description: Specifies the minimum
         number of seconds that can elapse between the time a password
         changes and when it can be changed again. A value of
         zero indicates that no delay is required between password
         updates.
```

7.341. minPasswdLen

```
   elementId: TBD
   name: minPasswdLen
   dataType: unsigned32
   status: current
   description: Specifies the minimum
         allowable password length. Valid values for this element are
         zero through PWLEN.
```

7.342. passwordHistLen

elementId: TBD
name: passwordHistLen
dataType: unsigned32
status: current
description: Specifies the length of
password history maintained. A new password cannot match any
of the previous usrmod0_password_hist_len passwords.
Valid values for this element are zero through DEF_MAX_PWHIST.

7.343. passwordComplexity

elementId: TBD
name: passwordComplexity
dataType: boolean
status: current
description: Indicates whether
passwords must meet the complexity requirements put forth
by the operating system.

7.344. reversibleEncryption

elementId: TBD
name: reversibleEncryption
dataType: boolean
status: current
description: Indicates whether
or not passwords are stored using reversible encryption.

7.345. portInfo

elementId: TBD
name: portInfo
dataType: list
structure: list (localAddress, localPort, transportProtocol,
pid, foreignAddress, foreignPort)
status: current
description: Information about open listening ports.

7.346. foreignPort

elementId: TBD
name: foreignPort
dataType: string
status: current
description: The TCP or UDP port to which
the program communicates.

7.347. printereffectiverights

elementId: TBD
name: printereffectiverights
dataType: list
structure: list (printerName, trusteeSid,
standardDelete, standardReadControl, standardWriteDac,
standardWriteOwner, standardSynchronize,
accessSystemSecurity, genericRead, genericWrite,
genericExecute, genericAll, printerAccessAdminister,
printerAccessUse, jobAccessAdminister, jobAccessRead)
status: current
description: Stores the effective rights of a printer that a discretionary access control list (DACL) structure grants to a specified trustee. The trustee’s effective rights are determined checking all access-allowed and access-denied access control entries (ACEs) in the DACL.

7.348. printerName

elementId: TBD
name: printerName
dataType: string
status: current
description: Specifies the name of the
printer.

7.349. printerAccessAdminister

elementId: TBD
name: printerAccessAdminister
dataType: boolean
status: current
description:

7.350. printerAccessUse

elementId: TBD
name: printerAccessUse
dataType: boolean
status: current
description:

7.351. jobAccessAdminister

```
   elementId: TBD
   name: jobAccessAdminister
   dataType: boolean
   status: current
   description:
```

7.352. jobAccessRead

```
   elementId: TBD
   name: jobAccessRead
   dataType: boolean
   status: current
   description:
```

7.353. registry

```
   elementId: TBD
   name: registry
   dataType: list
   structure: list (registryHive, registryKey, registryKeyName,
                    lastWriteTime, registryKeyType, registryKeyValue,
                    windowsView)
   status: current
   description: Specifies information that can be
         collected about a particular registry key.
```

7.354. registryHive

```
   elementId: TBD
   name: registryHive
   dataType: enumeration
   structure: HKEY_CLASSES_ROOT ; 0x1 ; This registry subtree
         contains information that associates file types with programs
         and configuration data for automation (e.g. COM
         objects and Visual Basic Programs).
         HKEY_CURRENT_CONFIG ; 0x2 ; This registry subtree contains
         configuration data for the current hardware profile.
         HKEY_CURRENT_USER ; 0x3 ; This registry subtree contains the
         user profile of the user that is currently logged into the
         system.
         HKEY_LOCAL_MACHINE ; 0x4 ; This registry subtree contains
         information about the local system.
         HKEY_USERS ; 0x5 ; This registry subtree contains user-specific
         data.
         ; 0x6 ; The empty string value is permitted here to allow
         for detailed error reporting.
   status: current
   description: The
         hive that the registry key belongs to.
```

7.355. registryKey

```
   elementId: TBD
   name: registryKey
   dataType: string
   status: current
   description: Describes the registry key.
         Note that the hive portion of the string should not be
         included, as this data can be found under the hive
         element.
```

7.356. registryKeyName

```
   elementId: TBD
   name: registryKeyName
   dataType: string
   status: current
   description: Describes the name of a
         registry key.
```

7.357. lastWriteTime

elementId: TBD
name: lastWriteTime
dataType: unsigned64
status: current
description: The last time that the key or any of its value entries were modified. The value of this entity represents the FILETIME structure which is a 64-bit value representing the number of 100-nanosecond intervals since January 1, 1601 (UTC). Last write time can be queried on any key, with hives being classified as a type of key. When collecting only information about a registry hive or key the last write time will be the time the key or any of its entries were modified. When collecting only information about a registry name the last write time will be the time the containing key was modified. Thus when collecting information about a registry name, the last write time does not correlate directly to the specified name. See the RegQueryInfoKey function lpftLastWriteTime.

7.358. registryKeyType

elementId: TBD
name: registryKeyType
dataType: enumeration
structure: reg_binary ; 0x1 ; The reg_binary type is used by registry keys that specify binary data in any form.
reg_dword ; 0x2 ; The reg_dword type is used by registry keys that specify an unsigned 32-bit integer.
reg_dword_little_endian ; 0x3 ; The reg_dword_little_endian type is used by registry keys that specify an unsigned 32-bit little-endian integer. It is designed to run on little-endian computer architectures.
reg_dword_big_endian ; 0x4 ; The reg_dword_big_endian type is used by registry keys that specify an unsigned 32-bit big-endian integer. It is designed to run on big-endian computer architectures.
reg_expand_sz ; 0x5 ; The reg_expand_sz type is used by registry keys to specify a null-terminated string that contains unexpanded references to environment variables (for example, "%PATH%").
reg_link ; 0x6 ; The reg_link type is used by the registry keys for null-terminated unicode strings. It is related to target path of a symbolic link created by the RegCreateKeyEx function.
reg_multi_sz ; 0x7 ; The reg_multi_sz type is used by registry keys that specify an array of null-terminated strings, terminated by two null characters.

reg_none; 0x8 ;
The reg_none type is used by registry keys that have no defined value type.
reg_qword; 0x9 ; The reg_qword type is used by registry keys that specify an unsigned 64-bit integer.
reg_qword_little_endian; 0xA ; The reg_qword_little_endian type is used by registry keys that specify an unsigned 64-bit integer in little-endian computer architectures.
reg_sz; 0xB ; The reg_sz type is used by registry keys that specify a single null-terminated string.
reg_resource_list; 0xC ; The reg_resource_list type is used by registry keys that specify a resource list.
reg_full_resource_descriptor; 0xD ; The reg_full_resource_descriptor type is used by registry keys that specify a full resource descriptor.
reg_resource_requirements_list; 0xE ; The reg_resource_requirements_list type is used by registry keys that specify a resource requirements list.
; 0xF ; The empty string value is permitted here to allow for detailed error reporting.

status: current

description:

Specifies the type of data stored by the registry key.

7.359. registryKeyValue

elementId: TBD
name: registryKeyValue
dataType: string
status: current
description: Holds the actual value
of the specified registry key. The representation of the value as well as the associated datatype attribute depends on type of data stored in the registry key. If the value being tested is of type REG_BINARY, then the datatype attribute should be set to ’binary’ and the data represented by the value entity should follow the xsd:hexBinary form. (each binary octet is encoded as two hex digits) If the value being tested is of type REG_DWORD, REG_QWORD, REG_DWORD_LITTLE_ENDIAN, REG_DWORD_BIG_ENDIAN, or REG_QWORD_LITTLE_ENDIAN then the datatype attribute should be set to ’int’ and the value entity should represent the data as an unsigned integer. DWORD and QWORD values represnt unsigned 32-bit and 64-bit integers, respectively. If the value being tested is of type REG_EXPAND_SZ, then the datatype attribute should be set to ’string’ and the pre-expanded string should be represented by the value entity. If the value being tested is of type REG_MULTI_SZ, then only a single string (one of the multiple strings) should be tested using the value entity with the datatype attribute set to ’string’. In order to test multiple values, multiple OVAL registry tests should be used. If the specified registry key is of type REG_SZ, then the datatype should be ’string’ and the value entity should be a copy of the string. If the value being tested is of type REG_LINK, then the datatype attribute should be set to ’string’ and the null-terminated Unicode string should be represented by the value entity.

7.360. regkeyauditedpermissions

```
elementId: TBD
name: regkeyauditedpermissions
dataType: list
structure: list (registryKey, trusteeSid, trusteeName,
      standardDelete, standardReadControl, standardWriteDac,
      standardWriteOwner, standardSynchronize,
      accessSystemSecurity, genericRead, genericWrite,
      genericExecute, genericAll, keyQueryValue, keySetValue,
      keyCreateSubKey, keyEnumerateSubKeys, keyNotify,
      keyCreateLink, keyWow6464Key, keyWow6432Key, keyWow64Res,
      windowsView)
status: current
description: Stores the audited access rights of a registry key
that a system access control list (SACL) structure grants to a
specified trustee. The trustee's audited access rights are
determined checking all access control entries (ACEs) in the SACL.
```

7.361. auditKeyQueryValue

```
elementId: TBD
name: auditKeyQueryValue
dataType: enumeration
structure: AUDIT_FAILURE ; 0x1 ; The audit type AUDIT_FAILURE is
used to perform audits on all unsuccessful occurrences of
specified events when auditing is enabled.
AUDIT_NONE ; 0x2 ; The audit type AUDIT_NONE is used to cancel
all auditing options for the specified events.
AUDIT_SUCCESS ; 0x3 ; The audit type AUDIT_SUCCESS is used to
perform audits on all successful occurrences of the specified
events when auditing is enabled.
AUDIT_SUCCESS_FAILURE ; 0x4 ; The audit type AUDIT_SUCCESS_FAILURE
is used to perform audits on all successful and unsuccessful
occurrences of the specified events when auditing is enabled.
; 0x5 ; The empty string value is permitted here to allow for
detailed error reporting.
status: current
description:
```

7.362. auditKeySetValue

elementId: TBD
name: auditKeySetValue
dataType: enumeration
structure: AUDIT_FAILURE ; 0x1 ; The audit type AUDIT_FAILURE is used to perform audits on all unsuccessful occurrences of specified events when auditing is enabled.
AUDIT_NONE ; 0x2 ; The audit type AUDIT_NONE is used to cancel all auditing options for the specified events.
AUDIT_SUCCESS ; 0x3 ; The audit type AUDIT_SUCCESS is used to perform audits on all successful occurrences of the specified events when auditing is enabled.
AUDIT_SUCCESS_FAILURE ; 0x4 ; The audit type AUDIT_SUCCESS_FAILURE is used to perform audits on all successful and unsuccessful occurrences of the specified events when auditing is enabled.
; 0x5 ; The empty string value is permitted here to allow for detailed error reporting.
status: current
description:

7.363. auditKeyCreateSubKey

elementId: TBD
name: auditKeyCreateSubKey
dataType: enumeration
structure: AUDIT_FAILURE ; 0x1 ; The audit type AUDIT_FAILURE is used to perform audits on all unsuccessful occurrences of specified events when auditing is enabled.
AUDIT_NONE ; 0x2 ; The audit type AUDIT_NONE is used to cancel all auditing options for the specified events.
AUDIT_SUCCESS ; 0x3 ; The audit type AUDIT_SUCCESS is used to perform audits on all successful occurrences of the specified events when auditing is enabled.
AUDIT_SUCCESS_FAILURE ; 0x4 ; The audit type AUDIT_SUCCESS_FAILURE is used to perform audits on all successful and unsuccessful occurrences of the specified events when auditing is enabled.
; 0x5 ; The empty string value is permitted here to allow for detailed error reporting.
status: current
description:

7.364. auditKeyEnumerateSubKeys

elementId: TBD
name: auditKeyEnumerateSubKeys
dataType: enumeration
structure: AUDIT_FAILURE ; 0x1 ; The audit type AUDIT_FAILURE is used to perform audits on all unsuccessful occurrences of specified events when auditing is enabled.
AUDIT_NONE ; 0x2 ; The audit type AUDIT_NONE is used to cancel all auditing options for the specified events.
AUDIT_SUCCESS ; 0x3 ; The audit type AUDIT_SUCCESS is used to perform audits on all successful occurrences of the specified events when auditing is enabled.
AUDIT_SUCCESS_FAILURE ; 0x4 ; The audit type AUDIT_SUCCESS_FAILURE is used to perform audits on all successful and unsuccessful occurrences of the specified events when auditing is enabled.
; 0x5 ; The empty string value is permitted here to allow for detailed error reporting.
status: current
description:

7.365. auditKeyNotify

elementId: TBD
name: auditKeyNotify
dataType: enumeration
structure: AUDIT_FAILURE ; 0x1 ; The audit type AUDIT_FAILURE is used to perform audits on all unsuccessful occurrences of specified events when auditing is enabled.
AUDIT_NONE ; 0x2 ; The audit type AUDIT_NONE is used to cancel all auditing options for the specified events.
AUDIT_SUCCESS ; 0x3 ; The audit type AUDIT_SUCCESS is used to perform audits on all successful occurrences of the specified events when auditing is enabled.
AUDIT_SUCCESS_FAILURE ; 0x4 ; The audit type AUDIT_SUCCESS_FAILURE is used to perform audits on all successful and unsuccessful occurrences of the specified events when auditing is enabled.
; 0x5 ; The empty string value is permitted here to allow for detailed error reporting.
status: current
description:

7.366. auditKeyCreateLink

elementId: TBD
name: auditKeyCreateLink
dataType: enumeration
structure: AUDIT_FAILURE ; 0x1 ; The audit type AUDIT_FAILURE is used to perform audits on all unsuccessful occurrences of specified events when auditing is enabled.
AUDIT_NONE ; 0x2 ; The audit type AUDIT_NONE is used to cancel all auditing options for the specified events.
AUDIT_SUCCESS ; 0x3 ; The audit type AUDIT_SUCCESS is used to perform audits on all successful occurrences of the specified events when auditing is enabled.
AUDIT_SUCCESS_FAILURE ; 0x4 ; The audit type AUDIT_SUCCESS_FAILURE is used to perform audits on all successful and unsuccessful occurrences of the specified events when auditing is enabled.
; 0x5 ; The empty string value is permitted here to allow for detailed error reporting.
status: current
description:

7.367. auditKeyWow6464Key

elementId: TBD
name: auditKeyWow6464Key
dataType: enumeration
structure: AUDIT_FAILURE ; 0x1 ; The audit type AUDIT_FAILURE is used to perform audits on all unsuccessful occurrences of specified events when auditing is enabled.
AUDIT_NONE ; 0x2 ; The audit type AUDIT_NONE is used to cancel all auditing options for the specified events.
AUDIT_SUCCESS ; 0x3 ; The audit type AUDIT_SUCCESS is used to perform audits on all successful occurrences of the specified events when auditing is enabled.
AUDIT_SUCCESS_FAILURE ; 0x4 ; The audit type AUDIT_SUCCESS_FAILURE is used to perform audits on all successful and unsuccessful occurrences of the specified events when auditing is enabled.
; 0x5 ; The empty string value is permitted here to allow for detailed error reporting.
status: current
description:

7.368. auditKeyWow6432Key

elementId: TBD
name: auditKeyWow6432Key
dataType: enumeration
structure: AUDIT_FAILURE ; 0x1 ; The audit type AUDIT_FAILURE is used to perform audits on all unsuccessful occurrences of specified events when auditing is enabled.
AUDIT_NONE ; 0x2 ; The audit type AUDIT_NONE is used to cancel all auditing options for the specified events.
AUDIT_SUCCESS ; 0x3 ; The audit type AUDIT_SUCCESS is used to perform audits on all successful occurrences of the specified events when auditing is enabled.
AUDIT_SUCCESS_FAILURE ; 0x4 ; The audit type AUDIT_SUCCESS_FAILURE is used to perform audits on all successful and unsuccessful occurrences of the specified events when auditing is enabled.
; 0x5 ; The empty string value is permitted here to allow for detailed error reporting.
status: current
description:

7.369. auditKeyWow64Res

elementId: TBD
name: auditKeyWow64Res
dataType: enumeration
structure: AUDIT_FAILURE ; 0x1 ; The audit type AUDIT_FAILURE is used to perform audits on all unsuccessful occurrences of specified events when auditing is enabled.
AUDIT_NONE ; 0x2 ; The audit type AUDIT_NONE is used to cancel all auditing options for the specified events.
AUDIT_SUCCESS ; 0x3 ; The audit type AUDIT_SUCCESS is used to perform audits on all successful occurrences of the specified events when auditing is enabled.
AUDIT_SUCCESS_FAILURE ; 0x4 ; The audit type AUDIT_SUCCESS_FAILURE is used to perform audits on all successful and unsuccessful occurrences of the specified events when auditing is enabled.
; 0x5 ; The empty string value is permitted here to allow for detailed error reporting.
status: current
description:

7.370. regkeyeffectiverights

```
   elementId: TBD
   name: regkeyeffectiverights
   dataType: list
   structure: list (registryHive, registryKey, trusteeSid,
         trusteeName, standardDelete, standardReadControl,
         standardWriteDac, standardWriteOwner, standardSynchronize,
         accessSystemSecurity, genericRead, genericWrite,
         genericExecute, genericAll, keyQueryValue, keySetValue,
         keyCreateSubKey, keyEnumerateSubKeys, keyNotify,
         keyCreateLink, keyWow6464Key, keyWow6432Key, keyWow64Res,
         windowsView)
   status: current
   description: Stores the effective rights of a registry key that a
   discretionary access control list (DACL) structure grants to a
   specified trustee. The trustee's effective rights are determined
   checking all access-allowed and access-denied access control
   entries (ACEs) in the DACL.
```

7.371. keyQueryValue

```
   elementId: TBD
   name: keyQueryValue
   dataType: boolean
   status: current
   description: Specifies whether or not
         permission is granted to query the key's value.
```

7.372. keySetValue

```
   elementId: TBD
   name: keySetValue
   dataType: boolean
   status: current
   description: Specifies whether or not
         permission is granted to set the key's value.
```

7.373. keyCreateSubKey

```
   elementId: TBD
   name: keyCreateSubKey
   dataType: boolean
   status: current
   description: Specifies whether or not
         permission is granted to create a subkey.
```

7.374. keyEnumerateSubKeys

elementId: TBD
name: keyEnumerateSubKeys
dataType: boolean
status: current
description: Specifies whether or
not permission is granted to list the subkeys associated
with key.

7.375. keyNotify

elementId: TBD
name: keyNotify
dataType: boolean
status: current
description:

7.376. keyCreateLink

elementId: TBD
name: keyCreateLink
dataType: boolean
status: current
description:

7.377. keyWow6464Key

elementId: TBD
name: keyWow6464Key
dataType: boolean
status: current
description:

7.378. keyWow6432Key

elementId: TBD
name: keyWow6432Key
dataType: boolean
status: current
description:

7.379. keyWow64Res

elementId: TBD
name: keyWow64Res
dataType: boolean
status: current
description:

7.380. service

elementId: TBD
name: service
dataType: list
structure: list (serviceName, displayName, description,
serviceType, startType, currentState, controlsAccepted,
startName, path, pid, serviceFlag, dependencies)
status: current
description: Stores information about Windows services that are
present on the system.

7.381. displayName

elementId: TBD
name: displayName
dataType: string
status: current
description: Specifies the name of the
service as specified in administrative tools.

7.382. description

elementId: TBD
name: description
dataType: string
status: current
description: Specifies the description of
the service.

7.383. serviceType

```
   elementId: TBD
   name: serviceType
   dataType: enumeration
   structure: SERVICE_FILE_SYSTEM_DRIVER ; 0x1 ; The
         SERVICE_FILE_SYSTEM_DRIVER type means that the service is
         a file system driver. The DWORD value that this
         corresponds to is 0x00000002.
         SERVICE_KERNEL_DRIVER ; 0x2 ; The SERVICE_KERNEL_DRIVER type
         means that the service is a driver. The DWORD value that
         this corresponds to is 0x00000001.
         SERVICE_WIN32_OWN_PROCESS ; 0x3 ; The SERVICE_WIN32_OWN_PROCESS
         type means that the service runs in its own process. The DWORD
         value that this corresponds to is 0x00000010.
         SERVICE_WIN32_SHARE_PROCESS ; 0x4 ; The
         SERVICE_WIN32_SHARE_PROCESS type means that the service runs
         in a process with other services. The DWORD value that this
         corresponds to is 0x00000020.
         SERVICE_INTERACTIVE_PROCESS ; 0x5 ; The
         SERVICE_WIN32_SHARE_PROCESS type means that the service runs
         in a process with other services. The DWORD value that this
         corresponds to is 0x00000100.
         ; 0x6 ; The empty string value is permitted here to allow for
         empty elements associated with error conditions.
   status: current
   description:
         Specifies the type of the service.
```

7.384. startType

elementId: TBD
name: startType
dataType: enumeration
structure: SERVICE_AUTO_START ; 0x1 ; The SERVICE_AUTO_START type means that the service is started automatically by the Service Control Manager (SCM) during startup. The DWORD value that this corresponds to is 0x00000002.
SERVICE_BOOT_START ; 0x2 ; The SERVICE_BOOT_START type means that the driver service is started by the system loader. The DWORD value that this corresponds to is 0x00000000.
SERVICE_DEMAND_START ; 0x3 ; The SERVICE_DEMAND_START type means that the service is started by the Service Control Manager (SCM) when StartService() is called. The DWORD value that this corresponds to is 0x00000003.
SERVICE_DISABLED ; 0x4 ; The SERVICE_DISABLED type means that the service cannot be started. The DWORD value that this corresponds to is 0x00000004.
SERVICE_SYSTEM_START ; 0x5 ; The SERVICE_SYSTEM_START type means that the service is a device driver started by IoInitSystem(). The DWORD value that this corresponds to is 0x00000001.
; 0x6 ; The empty string value is permitted here to allow for empty elements associated with error conditions.
status: current
description: Specifies when the service should be started.

7.385. currentState

```
   elementId: TBD
   name: currentState
   dataType: enumeration
   structure: SERVICE_CONTINUE_PENDING ; 0x1 ; The
         SERVICE_CONTINUE_PENDING type means that the service has been
         sent a command to continue, however, the command has
         not yet been executed. The DWORD value that this corresponds
         to is 0x00000005. SERVICE_PAUSE_PENDING ; 0x2 ; The
         SERVICE_PAUSE_PENDING type means that the service has been
         sent a command to pause, however, the command has not
         yet been executed. The DWORD value that this corresponds to
         is 0x00000006.
         SERVICE_PAUSED ; 0x3 ; The SERVICE_PAUSED type means that
         the service is paused. The DWORD value that this corresponds
         to is 0x00000007.
         SERVICE_RUNNING ; 0x4 ; The SERVICE_RUNNING type means that
         the service is running. The DWORD value that this
         corresponds to is 0x00000004.
         SERVICE_START_PENDING ; 0x5 ; The SERVICE_START_PENDING type
         means that the service has been sent a command to start,
         however, the command has not yet been executed. The DWORD
         value that this corresponds to is 0x00000002.
         SERVICE_STOP_PENDING ; 0x6 ; The SERVICE_STOP_PENDING type
         means that the service
         has been sent a command to stop, however, the command has
         not yet been executed. The DWORD value that this
         corresponds to is 0x00000003.
         SERVICE_STOPPED ; 0x7 ; The SERVICE_STOPPED type means that
         the service is stopped. The DWORD value that this corresponds
         to is 0x00000001.
         ; 0x8 ; The empty string value is permitted here to allow
         for empty elements associated with error conditions.
   status: current
   description: Specifies the current state of
         the service.
```

7.386. controlsAccepted

```
   elementId: TBD
   name: controlsAccepted
   dataType: enumeration
   structure:
     SERVICE_ACCEPT_NETBINDCHANGE ; 0x1 ;
     The SERVICE_ACCEPT_NETBINDCHANGE type means that the
     service is a network component and can accept changes in its
     binding without being stopped or restarted. The DWORD value
     that this corresponds to is 0x00000010.
     SERVICE_ACCEPT_PARAMCHANGE ; 0x2 ; The SERVICE_ACCEPT_PARAMCHANGE
```

```
      type means that the service can re-read its
      startup parameters without being stopped or restarted. The
      DWORD value that this corresponds to is 0x00000008.
      SERVICE_ACCEPT_PAUSE_CONTINUE ; 0x3 ; The
      SERVICE_ACCEPT_PAUSE_CONTINUE type means that the service
      can be paused or continued. The DWORD value that this
      corresponds to is 0x00000002.
      SERVICE_ACCEPT_PRESHUTDOWN ; 0x4 ; The
      SERVICE_ACCEPT_PRESHUTDOWN type means that the service can
      receive pre-shutdown notifications. The DWORD value
      that this corresponds to is 0x00000100.
      SERVICE_ACCEPT_SHUTDOWN ; 0x5 ; The SERVICE_ACCEPT_SHUTDOWN
      type means that the service can receive shutdown notifications.
      The DWORD value that this corresponds to is 0x00000004.
      SERVICE_ACCEPT_STOP ; 0x6 ; The SERVICE_ACCEPT_STOP type
      means that the service can be stopped. The DWORD value
      that this corresponds to is 0x00000001.
      SERVICE_ACCEPT_HARDWAREPROFILECHANGE ; 0x7 ; The
      SERVICE_ACCEPT_HARDWAREPROFILECHANGE type means that the
      service can receive notifications when the system's
      hardware profile changes. The DWORD value that this
      corresponds to is 0x00000020.
      SERVICE_ACCEPT_POWEREVENT ; 0x8 ; The SERVICE_ACCEPT_POWEREVENT
      type means that the service can receive notifications when the
      system's power status has changed. The DWORD value that this
      corresponds to is 0x00000040.
      SERVICE_ACCEPT_SESSIONCHANGE ; 0x9 ; The
      SERVICE_ACCEPT_SESSIONCHANGE type means that the service can
      receive notifications when the system's session
      status has changed. The DWORD value that this corresponds
      to is 0x00000080.
      SERVICE_ACCEPT_TIMECHANGE ; 0xA ; The SERVICE_ACCEPT_TIMECHANGE
      type means that the service can receive notifications when
      the system time changes. The DWORD value that this corresponds
      to is 0x00000200.
      SERVICE_ACCEPT_TRIGGEREVENT ; 0xB ; The
      SERVICE_ACCEPT_TRIGGEREVENT type means that the service can
      receive notifications when an event that the service
      has registered for occurs on the system. The DWORD value that
      this corresponds to is 0x00000400.
      ; 0xC ; The empty string value is permitted here to allow
      for empty elements associated with error conditions.
   status: current

   description: Specifies the control codes that a service will
                accept and process.
```

7.387. startName

elementId: TBD
name: startName
dataType: string
status: current
description: Specifies the account under which the process should run.

7.388. serviceFlag

elementId: TBD
name: serviceFlag
dataType: boolean
status: current
description: Specifies whether the service is in a system process that must always run (true) or if the service is in a non-system process or is not running (false).

7.389. dependencies

elementId: TBD
name: dependencies
dataType: string
status: current
description: Specifies the dependencies of this service on other services.

7.390. serviceeffectiverights

elementId: TBD
name: serviceeffectiverights
dataType: list
structure: list (serviceName, trusteeSid, standardDelete, standardReadControl, standardWriteDac, standardWriteOwner, genericRead, genericWrite, genericExecute, serviceQueryConf, serviceChangeConf, serviceQueryStat, serviceEnumDependents, serviceStart, serviceStop, servicePause, serviceInterrogate, serviceUserDefined)
status: current
description: Stores the effective rights of a service that a discretionary access control list (DACL) structure grants to a specified trustee. The trustee’s effective rights are determined by checking all access-allowed and access-denied access control entries (ACEs) in the DACL.

7.391. trusteeSid

elementId: TBD
name: trusteeSid
dataType: string
status: current
description: Specifies the SID that is
associated with a user, group, system, or program (such as a
Windows service).

7.392. serviceQueryConf

elementId: TBD
name: serviceQueryConf
dataType: boolean
status: current
description: Specifies whether or
not permission is granted to query the service configuration.

7.393. serviceChangeConf

elementId: TBD
name: serviceChangeConf
dataType: boolean
status: current
description: Specifies whether or
not permission is granted to change service configuration.

7.394. serviceQueryStat

elementId: TBD
name: serviceQueryStat
dataType: boolean
status: current
description: Specifies whether or
not permission is granted to query the service control
manager about the status of the service.

7.395. serviceEnumDependents

elementId: TBD
name: serviceEnumDependents
dataType: boolean
status: current
description: Specifies whether
or not permission is granted to query for an enumeration of
all the services dependent on the service.

7.396. serviceStart

elementId: TBD
name: serviceStart
dataType: boolean
status: current
description: Specifies whether or not
permission is granted to start the service.

7.397. serviceStop

elementId: TBD
name: serviceStop
dataType: boolean
status: current
description: Specifies whether or not
permission is granted to stop the service.

7.398. servicePause

elementId: TBD
name: servicePause
dataType: boolean
status: current
description: Specifies whether or not
permission is granted to pause or continue the service.

7.399. serviceInterrogate

elementId: TBD
name: serviceInterrogate
dataType: boolean
status: current
description: Specifies whether or not permission is granted to
request the service to report its status immediately.

7.400. serviceUserDefined

elementId: TBD
name: serviceUserDefined
dataType: boolean
status: current
description: Specifies whether or
not permission is granted to specify a user-defined
control code.

7.401. sharedresourceauditedpermissions

elementId: TBD
name: sharedresourceauditedpermissions
dataType: list
structure: list (netname, trusteeSid,
standardDelete, standardReadControl, standardWriteDac,
standardWriteOwner, standardSynchronize,
accessSystemSecurity, genericRead, genericWrite,
genericExecute, genericAll)
status: current
description: Stores
the audited access rights of a shared resource that a system access control list (SACL) structure grants to a specified trustee. The trustee’s audited access rights are determined checking all access control entries (ACEs) in the SACL.

7.402. netname

elementId: TBD
name: netname
dataType: string
status: current
description: Specifies the name associated
with a particular shared resource.

7.403. sharedresourceeffectiverights

elementId: TBD
name: sharedresourceeffectiverights
dataType: list
structure: list (netname, trusteeSid,
standardDelete, standardReadControl, standardWriteDac,
standardWriteOwner, standardSynchronize,
accessSystemSecurity, genericRead, genericWrite,
genericExecute, genericAll)
status: current
description: Stores
the effective rights of a shared resource that a discretionary access control list (DACL) structure grants to a specified trustee. The trustee’s effective rights are determined checking all access-allowed and access-denied access control entries (ACEs) in the DACL.

7.404. user

elementId: TBD
name: user
dataType: list
structure: list (username, enabled, group, lastLogon)
status: current
description: Specifies the groups to which a user belongs.

7.405. enabled

elementId: TBD
name: enabled
dataType: boolean
status: current
description: Represents whether the
particular user is enabled or not.

7.406. lastLogon

elementId: TBD
name: lastLogon
dataType: unsigned32
status: current
description: The date and time when the
last logon occurred.

7.407. groupSid

elementId: TBD
name: groupSid
dataType: string
status: current
description: Represents the SID of a
particular group. If the specified user belongs to more than
one group, then multiple groupSid elements are
applicable. If the specified user is not a member of a single
group, then a single groupSid element should be
incldued with a status of 'does not exist'. If there is an
error determining the groups that the user belongs to,
then a single groupSid element should be included with a
status of 'error'.

7.408. endpointType

```
elementId: TBD
name: endpointType
dataType: enumeration
status: current
description: The possible types of endpoint in the
             enterprise.
structure:
workstation; 0x1; Workstation Endpoint
printer;     0x2; Printer Endpoint
router;      0x3; Router Endpoint
tablet;      0x4; Tablet Endpoint
```

7.409. endpointPurpose

```
elementId: TBD
name: endpointPurpose
dataType: string
status: current
description: A description of how the endpoint is
             used within the enterprise.
             Examples include end user system,
             and public web server.
```

7.410. endpointCriticality

```
elementId: TBD
name: endpointCriticality
dataType: string
status: current
description: An enterprise-defined rating which
             indicates the criticality of the
             endpoint. The rating should be
             specific enough to assess the impact
             to the overall enterprise if the
             endpoint is attacked or lost.
```

7.411. ingestTimestamp

```
elementId: TBD
name: ingestTimestamp
dataType: dateTimeSeconds
status: current
description: The point in time that the
             description of a vulnerability was
             received by the enterprise.
```

7.412. vulnerabilityVersion

```
elementId: TBD
name: vulnerabilityVersion
dataType: string
status: current
description: The version or iteration of the
             vulnerability description information
             (reported by the author, if applicable).
```

7.413. vulnerabilityExternalId

```
elementId: TBD
name: vulnerabilityExternalId
dataType: string
status: current
description: An external or third-party ID
             assigned to the vulnerability
             description. This could be multiple
             IDs in some cases (e.g., vendor bug
             ID, global ID, discoverer's local ID,
             third-party vulnerability database
             ID, etc.).
```

7.414. vulnerabilitySeverity

```
elementId: TBD
name: vulnerabilitySeverity
dataType: string
status: current
description: The severity of the vulnerability
             (reported by the author, if applicable).
```

7.415. assessmentTimestamp

```
elementId: TBD
name: assessmentTimestamp
dataType: dateTimeSeconds
status: current
description: The point in time that the assessment
             was performed against an endpoint.
```

7.416. vulnerableSoftware

```
elementId: TBD
name: vulnerableSoftware
dataType: list
status: current
description: A listing of software products
             installed on the endpoint which are
             known to have vulnerabilities.
structure: list(softwareInstance*)
```

7.417. endpointVulnerabilityStatus

```
elementId: TBD
name: endpointVulnerabilityStatus
dataType: enumeration
status: current
description: Overall vulnerability status of an
             enterprise endpoint.
structure: Pass; 0x1; Endpoint passed the
                      vulnerability test(s).
           Fail; 0x2; Endpoint failed the
                      vulnerability test(s).
```

7.418. vulnerabilityDescription

```
elementId: TBD
name: vulnerabilityDescription
dataType: string
status: current
description: A human-readable description of the
             vulnerability.
```

8. Acknowledgements

Many of the specifications in this document have been developed in a public-private partnership with vendors and end-users. The hard work of the SCAP community is appreciated in advancing these efforts to their current level of adoption.

Over the course of developing the initial draft, Brant Cheikes, Matt Hansbury, Daniel Haynes, Scott Pope, Charles Schmidt, and Steve Venema have contributed text to many sections of this document.

9. IANA Considerations

This document specifies an initial set of Information Elements for SACM in Section 7. An Internet Assigned Numbers Authority (IANA) registry will be created and populated with the Information Elements in Section 7. New assignments for SACM Information Elements will be administered by IANA through Expert Review [RFC2434]. The designated experts MUST check the requested Information Elements for completeness and accuracy of the submission with respect to the template and requirements expressed in Section 4 and Section 4.1. Requests for Information Elements that duplicate the functionality of existing Information Elements SHOULD be declined. The smallest available Information Element identifier SHOULD be assigned to a new Information Element. The definition of new Information Elements MUST be published using a well-established and persistent publication medium.

10. Security Considerations

Posture Assessments need to be performed in a safe and secure manner. In that regard, there are multiple aspects of security that apply to the communications between components as well as the capabilities themselves. This information model only contains an initial listing of items that need to be considered with respect to security and will need to be augmented as the model continues to be developed.

Security considerations include:

Authentication: Every SACM Component and asset needs to be able to identify itself and verify the identity of other SACM Components and assets.

Confidentiality: Communications between SACM Components need to be protected from eavesdropping or unauthorized collection. Some communications between SACM Components and assets may need to be protected as well.

Integrity: The information exchanged between SACM Components needs to be protected from modification. Some exchanges between assets and SACM Components will also have this requirement.

Restricted Access: Access to the information collected, evaluated, reported, and stored should only be viewable and consumable to authenticated and authorized entities.

Considerations with respect to the operational aspects of collection, evaluation, and storage security automation information can be found in Section 11.

Considerations concerning the privacy of security automation information can be found in Section 12.

11. Operational Considerations

The following sections outline a series of operational considerations for SACM deployments within an organization. This section may be expanded to include other considerations as the WG gains additional operational experience with SACM deployments and extending the information model.

11.1. Endpoint Designation

In order to successfully carry out endpoint posture assessment, it is necessary to be able to identify the endpoints on a network and track the changes to them over time. Specifically, enabling SACM Components to:

- Tell whether two endpoint attribute assertions concern the same endpoint
- Respond to compliance measurements, for example by reporting, remediating, and quarantining (SACM does not specify these responses, but SACM exists to enable them).

Ideally, every endpoint would be identified by a unique identifier present on the endpoint, but, this is complicated due to different factors such as the variety of endpoints on a network, the ability of tools to reliably access such an identifer, and the ability of tools to correlate disparate identifiers. As a result, it is necessary for an endpoint to be identified by a set of attributes that uniquely identify it on a network. The set of attributes that uniquely identify an endpoint on a network will likely vary by organization; however, there are a number of properties to consider when selecting identifying attributes as some are better suited for identification purposes than others.

Multiplicity: Is the attribute typically associated with a single endpoint or with multiple endpoints? If the attribute is associated with a single endpoint, it is better for identifying an endpoint on a network.

Persistence: How likely is the attribute to change? Does it never change? Does it only change when the endpoint is reprovisioned? Does it only change due to an event? Does it change on an ad-hoc and often unpredictable basis? Does it constantly change? The less likely it is for an attribute to

change over time, the better it is for identifying an endpoint on a network.

Immutability: How difficult is it to change the attribute? Is the attribute hardware rooted and never changes? Can the attribute be changed by a user/process with the appropriate access? Can the attribute be changed without controlled access. The less likely an attribute is to change over time, the better chance it will be usable to identify an endpoint over time.

Verifiable: Can the attribute be corroborated? Can the attribute be externally verified with source authentication? Can the attribute be externally verified without source authentication? Is it impossible to externally verify the attribute. Attributes that can be externally verified are more likely to be accurate and are better for identifying endpoints on a network.

With that said, requiring SACM Components and end users to constantly refer to a set of attributes to identify an endpoint, is particularly burdensome. As a result, SACM supports the concept of a target endpoint label which associates an identifier (unique to a SACM domain) with the set of attributes used by an organization to identify endpoints on a network. Once defined for an endpoint, the target endpoint label can be used in place of the set of identifying attributes.

11.2. Timestamp Accuracy

An organization will likely have different collectors deployed across the network that will be configured to collect posture attributes on varying frequencies (periodic, ad-hoc, event-driven, on endpoint, off endpoint, etc.). Some collectors will detect changes as soon as they occur whereas others will detect them at a later point during a periodic scan or when an event has triggered the collection of posture attributes. Furthermore, some changes will be detected on the endpoint and others will be observed off of the endpoint. As a result of these differences, the accuracy of the timestamp associated with the collected information will vary. For example, if a collector is only running once every 12 hours, the change probably happened at some point in time prior to the scan and the timestamp is likely not accurate. Due to this, it is important for system administrators to determine if the accuracy of a timestamp is good enough for their intended purposes.

12. Privacy Considerations

In the IETF, there are privacy concerns with respect to endpoint identity and monitoring. This is especially true when the activity on an endpoint can be linked to a particular person. For example, by correlating endpoint attributes such as usernames, certificates, etc. with browser activity, it may be possible to gain insight in to user behavior and trends beyond what is required to carry out endpoint posture assessments. In the hands of the wrong person, this information could be used to negatively influence a user's behavior or to plan attacks against the organization's infrastructure.

As a result, SACM data models should incorporate a mechanism by which an organization can designate which endpoint attributes are considered sensitive with respect to privacy. This will allow SACM Components to handle endpoint attributes in a manner consistent with the organization's privacy policies. Furthermore, organization's should put the proper mechanism in place to ensure endpoint attributes are protected when transmitted, stored, and accessed to ensure only authorized parties are granted access.

It should also be noted that some of this is often mitigated by organizational policies that require a user of an organization's network to consent to some level of monitoring in return for access to the network and other resources. The information that is monitored and collected will vary by organization and further highlights the need for a mechanism by which an organization can specify what constitutes privacy sensitive information for them.

13. References

13.1. Normative References

[PEN] Internet Assigned Numbers Authority, "Private Enterprise Numbers", July 2016, <https://www.iana.org/assignments/enterprise-numbers/enterprise-numbers>.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <http://www.rfc-editor.org/info/rfc2119>.

13.2. Informative References

[I-D.ietf-sacm-requirements] Cam-Winget, N. and L. Lorenzin, "Secure Automation and Continuous Monitoring (SACM) Requirements", draft-ietf-sacm-requirements-01 (work in progress), October 2014.

[I-D.ietf-sacm-terminology]
Waltermire, D., Montville, A., Harrington, D., and N. Cam-Winget, "Terminology for Security Assessment", draft-ietf-sacm-terminology-05 (work in progress), August 2014.

[RFC2434] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", RFC 2434, DOI 10.17487/RFC2434, October 1998, <http://www.rfc-editor.org/info/rfc2434>.

[RFC3580] Congdon, P., Aboba, B., Smith, A., Zorn, G., and J. Roese, "IEEE 802.1X Remote Authentication Dial In User Service (RADIUS) Usage Guidelines", RFC 3580, DOI 10.17487/RFC3580, September 2003, <http://www.rfc-editor.org/info/rfc3580>.

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

[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, <http://www.rfc-editor.org/info/rfc5209>.

[RFC7012] Claise, B., Ed. and B. Trammell, Ed., "Information Model for IP Flow Information Export (IPFIX)", RFC 7012, DOI 10.17487/RFC7012, September 2013, <http://www.rfc-editor.org/info/rfc7012>.

[RFC7632] Waltermire, D. and D. Harrington, "Endpoint Security Posture Assessment: Enterprise Use Cases", RFC 7632, DOI 10.17487/RFC7632, September 2015, <http://www.rfc-editor.org/info/rfc7632>.

Appendix A. Change Log

A.1. Changes in Revision 01

Added some proposed normative text.

For provenance:

- Added a class "Method"

- Added the produced-using relationship between an AVP and a method

Added the produced-by relationship between a Guidance and a SACM Component

Added the hosted-by relationship between a SACM Component and an Endpoint

asserted-by and summarized-by have been renamed to produced-by.

"User" is now "Account". If a user has different credentials, SACM cannot know that they belong to the same user. But, per Kim W, many organizations do have accounts that associate credentials.

The multiplicity of the based-on relationships has been corrected.

More relationships now have labels, per UML convention.

The diagram no longer has causal arrow. They had become redundant and were nonstandard and clutter.

Renamed "credential" to "identity", following industry usage. A credential includes proof, such as a key or password. A username or a distinguished name is called an "identity".

Removed Session, because an endpoint’s network activity is not SACM’s initial focus

Removed Authorization, for the same reason

Added many-to-many relationship between Hardware Component and Endpoint, for clarity

Added many-to-many relationship between Software Component and Endpoint, for clarity

Added "contains" relationship between Network Interface and Network Interface

Removed relationship between Network Interface and Account. The endpoint knows the identity it used to gain network access. The PDP also knows that. But they probably do not know the account.

Added relationship between Network Interface and Identity. The endpoint and the PDP will typically know the identity.

Made identity-to-account a many-to-one relationship.

A.2. Changes in Revision 02

Added Section Identifying Attributes.

Split the figure into Figure Model of Endpoint and Figure Information Elements.

Added Figure Information Elements Take 2, proposing a triple-store model.

Some editorial cleanup

A.3. Changes in Revision 03

Moved Appendix A.1, Appendix A.2, and Mapping to SACM Use Cases into the Appendix. Added a reference to it in Section 1

Added the Section 4 section. Provided notes for the type of information we need to add in this section.

Added the Section 6 section. Moved sections on Endpoint, Hardware Component, Software Component, Hardware Instance, and Software Instance there. Provided notes for the type of information we need to add in this section.

Removed the Provenance of Information Section. SACM is not going to solve provenance rather give organizations enough information to figure it out.

Updated references to the Endpoint Security Posture Assessment: Enterprise Use Cases document to reflect that it was published as an RFC.

Fixed the formatting of a few figures.

Included references to [RFC3580] where RADIUS is mentioned.

A.4. Changes in Revision 04

Integrated the IPFIX [RFC7012] syntax into Section 4.

Converted many of the existing SACM Information Elements to the IPFIX syntax.

Included existing IPFIX Information Elements and datatypes that could likely be reused for SACM in Section 7 and Section 4 respectively.

Removed the sections related to reports as described in https://github.com/sacmwg/draft-ietf-sacm-information-model/issues/30.

Cleaned up other text throughout the document.

A.5. Changes in Revision 05

Merged proposed changes from the I-D IM into the WG IM (https://github.com/sacmwg/draft-ietf-sacm-information-model/issues/41).

Fixed some formatting warnings.

Removed a duplicate IE and added a few IE datatypes that were missing.

A.6. Changes in Revision 06

Clarified that the SACM statement and content-element subjects are conceptual and that they do not need to be explicitly defined in a data model as long as the necessary information is provided.

Updated the IPFIX syntax used to define Information Elements. There are still a couple of open issues that need to be resolved.

Updated some of the Information Elements contained in Section 7 to use the revised IPFIX syntax. The rest of the Information Elements will be converted in a later revision.

Performed various clean-up and refactoring in Sections 6 and 7. Still need to go through Section 8.

Removed appendices that were not referenced in the body of the draft. The text from them is still available in previous revisions of this document if needed.

A.7. Changes in Revision 07

Made various changes to the IPFIX syntax based on discussions at the IETF 96 Meeting. Changes included the addition of a structure property to the IE specification template, the creation of an enumeration datatype, and the specification of an IE naming convention.

Provided text to define Collection Guidance, Evaluation Guidance, Classification Guidance, Storage Guidance, and Evaluation Results.

Included additional IEs related to software, configuration, and the vulnerability assessment scenario.

Added text for the IANA considerations, security considerations, operational considerations, and privacy considerations sections.

Performed various other editorial changes and clean-up.

A.8. Changes in Revision 08

Clarified text that describes subjects and attributes.

Clarified text that describes SACM Statements and Content Elements.

Removed stray metadata property fields from the definitions of several IEs.

Specified a syntax for defining category IEs.

Added an anyCategory IE that represents any IE in the IM.

Fixed several errors reported by the Travis-CI continuous integration service.

Performed various other editorial changes and clean-up.

A.9. Changes in Revision 09

Added "derived", "authority", and "verified" to the collectionTaskType IE (https://github.com/sacmwg/draft-ietf-sacm-information-model/issues/18).

Updated IE examples that use content-type to use statement-type (https://github.com/sacmwg/draft-ietf-sacm-information-model/issues/56).

Added "networkZoneLocation", "layer2NetworkLocation", and "layer3NetworkLocation" IEs (https://github.com/sacmwg/draft-ietf-sacm-information-model/issues/9).

Created a softwareClass attribute IE and added it to the softwareInstance subject IE. Also, removed the os* attribute IEs (https://github.com/sacmwg/draft-ietf-sacm-information-model/issues/10).

A.10. Changes in Revision 10

Added several IEs necessary for the SACM Vulnerability Assessment Scenario (https://github.com/sacmwg/draft-ietf-sacm-information-model/issues/43).

Fixed various typos and formatting issues.

Authors' Addresses

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

Email: david.waltermire@nist.gov

Kim Watson
United States Department of Homeland Security
DHS/CS&C/FNR
245 Murray Ln. SW, Bldg 410
MS0613
Washington, DC 20528
USA

Email: kimberly.watson@hq.dhs.gov

Clifford Kahn
Pulse Secure, LLC
2700 Zanker Road, Suite 200
San Jose, CA 95134
USA

Email: cliffordk@pulsesecure.net

Lisa Lorenzin
Pulse Secure, LLC
2700 Zanker Road, Suite 200
San Jose, CA 95134
USA

Email: llorenzin@pulsesecure.net

Michael Cokus
The MITRE Corporation
903 Enterprise Parkway, Suite 200
Hampton, VA 23666
USA

Email: msc@mitre.org

Daniel Haynes
The MITRE Corporation
202 Burlington Road
Bedford, MA 01730
USA

Email: dhaynes@mitre.org

Henk Birkholz
Fraunhofer SIT
Rheinstrasse 75
Darmstadt 64295
Germany

Email: henk.birkholz@sit.fraunhofer.de

SACM Working Group
Internet-Draft
Intended status: Informational
Expires: June 17, 2019

H. Birkholz
Fraunhofer SIT
J. Lu
Oracle Corporation
J. Strassner
Huawei Technologies
N. Cam-Winget
Cisco Systems
A. Montville
CIS

December 14, 2018

Security Automation and Continuous Monitoring (SACM) Terminology
draft-ietf-sacm-terminology-16

Abstract

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

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at https://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on June 17, 2019.

Copyright Notice

Table of Contents

1. Introduction

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

2. Terms and Definitions

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

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

In the context of SACM, an assertion is the output of a SACM Component in the form of a SACM Statement (including metadata about the data source and data origin, e.g. timestamps). While the validity of an assertion about Content and Content Metadata cannot be verified without, for example, Integrity Proofing of the

Data Source, an assertion (and therefore a SACM statement, respectively) of the validity of Statement Metadata can by enabled by including corresponding Integrity Evidence created by the Data Origin.

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

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, the basic composite definitions that are defined in the SACM Information Model.

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

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

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

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

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

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

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

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

ad-hoc, e.g. triggered by a unsolicited query

conditional, e.g. triggered in accordance with policies included in the compositions of workflows

scheduled, e.g. in regular intervals, such as every minute or weekly

continuously, e.g. a network behavior observation

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

Self-Reporting: A SACM component located on the target endpoint itself conducts the collection task.

Remote-Acquisition: A SACM component located on an Endpoint different from the target endpoint conducts the collection task via interfaces available on the target endpoint, e.g. SNMP/ NETCONF or WMI.

Behavior-Observation: A SACM component located on an Endpoint different from the target endpoint observes network traffic related to the target endpoint and conducts the collection task via interpretation of that network traffic.

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

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

Configuration: A non-volatile subset of the endpoint attributes of 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.

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

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

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

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

Content Element: Content elements constitute the payload data (SACM content) transferred via statement Subjects emitted by providers of information. Every content element Subject includes a specific content Subject and a corresponding content metadata Subject.

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

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

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

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

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

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

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

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

Data At Rest: Data that is stored.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Endpoint Attribute: Is a discreet endpoint characteristic that is computably observable.

Endpoint Attributes typically constitute Attributes that can be bundled into Subject (e.g. information about a specific network interface can be represented via a set of multiple AVP).

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

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

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

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

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

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

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

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

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

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

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

Expected Endpoint Attribute State: The policy-compliant state of an endpoint attribute that is to be compared against.

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

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

Examples of such processes/tasks include automated device management, remediation, collection, evaluation. Guidance influences the behavior of a SACM Component and is considered content of the management plane. In the context of SACM, guidance is machine-readable and can be manually or automatically generated

or provided. Typically, the tasks that provide guidance to SACM components have a low-frequency and tend to be sporadic.

There are two types of guidance:

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

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

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

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

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

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

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

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

information element in the SACM information model has a unique name. Endpoint attributes or time stamps, for example, are represented as information elements in the SACM information model.

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

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

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

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

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

Management Plane: An architectural component providing common functions to steer the behavior of SACM components, e.g. their behavior on the control plane.

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

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

The following characteristics are a summery derived from the Common Information Model and ITU-T X.213. Each Network Interface of a specific layer can be associated with one or more addresses appropriate for that layer. There is no guarantee that a network

address is globally unique. A dedicated authority entity can provide a level of assurance that a network address is unique in its given scope. In essence, there is always a scope to a network address, in which it is intended to be unique.

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

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

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

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

Metadata: Data about data.

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

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

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

addresses). This information may constitute one or more posture attributes.

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

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

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

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

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

SACM Broker Controller: A SACM Broker Controller is a controller that contains control plane functions to provide and/or connect services on behalf of other SACM components via interfaces on the control plane.

A broker may provide, for example, authorization services and find, upon request, SACM components providing requested services.

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

In the context of SACM, a set of SACM functions composes a SACM component. A SACM component conducts SACM tasks, acting on control plane, data plane and/or management plane via corresponding SACM interfaces. SACM defines a set of standard components (e.g. a collector, a broker, or a data store). A SACM component contains at least a basic set of control plane functions and can contain data plane and management plane functions. A SACM component residing on an endpoint assigns one or more SACM roles

to the corresponding endpoint due to the SACM functions it is composed of. A SACM component "resides on" an endpoint and an endpoint "contains" a SACM component, correspondingly. For example, a SACM component that is composed solely of functions that provide information would only take on the role of a provider.

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

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

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

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

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

SACM content includes the SACM data models.

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

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

SACM Function: A behavioral aspect of a SACM component that provides external SACM Interfaces or internal interfaces to other SACM Functionse.

For example, a SACM Function with SACM Interfaces on the Control Plane can provide a brokering function to other SACM Components. Via Data Plane interfaces, a SACM Function can act as a provider and/or as a consumer of information. SACM Functions can be propagated as the Capabilities of a SACM Component and can be discovered by or negotiated with other SACM Components.

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

[I-D.ietf-i2nsf-terminology] defines interface as a "set of operations one object knows it can invoke on, and expose to, another object," and further defines interface by stating that an interface "decouples the implementation of the operation from its

specification. An interface is a subset of all operations that a given object implements. The same object may have multiple types of interfaces to serve different purposes."

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

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

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

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

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

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

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

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

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

Software Component: A software package installed on an endpoint.

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

Software Instance: A running instance of a software component.

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

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

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

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

Statement: A statement is the root/top-level subject defined in the SACM information model.

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

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

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

In contrast to the definitions of subject provided by [RFC4949], a subject in the scope of SACM is neither "a system entity that

causes information to flow among objects or changes the system state" nor "a name of a system entity that is bound to the data items in a digital certificate".

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

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

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

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

Every endpoint that is not specifically designated as an excluded endpoint is a target endpoint. A target endpoint is not part of a SACM domain unless it contains a SACM component (e.g. a SACM component that publishes collection results coming from an internal collector).

A target endpoint is similar to a device that is a Target of Evaluation (TOE) as defined in Common Criteria and as referenced by {{RFC4949}.

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

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

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

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

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

A characterization record is intended to be a representation of an endpoint. It cannot be assured that a record distinctly represents a single target endpoint unless a set of one or more endpoint attributes that compose a unique set of identifying endpoint attributes are included in the record. Otherwise, the set of identifying attributes included in a record can match more than one target endpoints, which are - in consequence - indistinguishable to a SACM domain until more qualifying endpoint attributes can be acquired and added to the record. A characterization record is maintained over time in order to assert that acquired endpoint attributes are either about an endpoint that was encountered before or an endpoint that has not been encountered before in a SACM domain. A characterization record can include, for example, acquired configuration, state or observed behavior of a specific target endpoint. Multiple and even conflicting instances of this information can be included in a characterization record by using timestamps and/or data origins to differentiate them. The endpoint attributes included in a characterization record can be used to re-identify a distinct target endpoint over time. Classes or profiles can be associated with a characterization record via the Classification Task in order to guide collection, evaluation or remediation tasks.

Target Endpoint Characterization Task: An ongoing task of continuously adding acquired endpoint attributes to a corresponding record. The TE characterization task manages the representation of encountered target endpoints in the SACM domain in the form of characterization records. For example, the output of a target endpoint discovery task or a collection task can be processed by the characterization task and added to the record. The TE characterization Task also manages these representations of target endpoints encountered in the SACM domain by splitting or merging the corresponding records as new or more refined endpoint attributes become available.

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

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

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

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

Target Endpoint Profile: A bundle of expected or desired component composition, configurations and states that is associated with a target endpoint.

The corresponding task by which the association with a target endpoint takes places is the endpoint classification task. The task by which an endpoint profile is created is the endpoint characterization task. A type or class of target endpoints can be defined via a target endpoint profile. Examples include: printers, smartphones, or an office PC.

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

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

A SACM task can be triggered by other operations or functions (e.g. a query from another SACM component or an unsolicited push on the data plane due to an ongoing subscription). A task is part of a SACM process chain. A task starts at a given point in time

and ends in a deterministic state. With the exception of a collection task, a SACM task consumes SACM statements provided by other SACM components. The output of a task is a result that can be provided (e.g. published) on the data plane.

The following tasks are defined by SACM:

Target Endpoint Discovery

Target Endpoint Characterization

Target Endpoint Classification

Collection

Evaluation [TBD]

Information Sharing [TBD]

SACM Component Discovery

SACM Component Authentication [TBD]

SACM Component Authorization [TBD]

SACM Component Registration [TBD]

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

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

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

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

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

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

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

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

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

Vulnerability assessment capabilities: An assessment capability that is tailored toward determining whether a set of endpoints is vulnerable according to vulnerability description information.

Workflow: A workflow is a modular composition of tasks that can contain loops, conditionals, multiple starting points and multiple endpoints.

The most prominent workflow in SACM is the assessment workflow.

3. IANA Considerations

This memo includes no request to IANA.

4. Security Considerations

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

5. Acknowledgements

6. Change Log

Changes from version 00 to version 01:

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

Changes from version 01 to version 02:

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

Changes from version 02 to version 03:

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

Changes from version 03 to version 04:

- TODO

Changes from version 04 to version 05:

- TODO

Changes from version 05 to version 06:

- Updated author information.
- Combined "Pre-defined Terms" with "New Terms and Definitions".
- Removed "Requirements language".
- Removed unused reference to use case draft; resulted in removal of normative references.
- Removed introductory text from Section 1 indicating that this document is intended to be temporary.
- Added placeholders for missing change log entries.

Changes from version 06 to version 07:

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

Changes from version 07 to version 08:

- Added Assertion, Collection Result, Collector, Excluded Endpoint, Internal Collector, Network Address, Network Interface, SACM Domain, Statement, Target Endpoint Identifier, Target Endpoint Label, Timestamp.
- Major updates to Attributes, Broker, Collection Task, Consumer, Controller, Control Plane, Endpoint Attributes, Expected Endpoint State, SACM Function, Provider, Proxy, Repository, SACM Role, Target Endpoint.
- Minor updates to Asset, Building Block, Data Origin, Data Source, Data Provenance, Endpoint, Management Plane, Posture, Posture Attribute, SACM Component, SACM Component Discovery, Target Endpoint Discovery.

- Relabeled Function to SACM Function.

Changes from version 08 to version 09:

- Updated author list.
- Added Data Plane, Endpoint Characterization, Endpoint Classification, Guidance, Interaction Model, Software Component, Software Instance, Software Package, Statement, Target Endpoint Profile, SACM Task.
- Removed Building Block.
- Major updates to Control Plane, Endpoint Attribute, Expected Endpoint State, Information Model, Management Plane.
- Minor updates to Attribute, Capabilities, SACM Function, SACM Component, Collection Task.
- Moved Asset Characterization to The Attic.

Changes from version 09 to version 10:

- Added Configuration Drift, Data in Motion, Data at Rest, Endpoint Management Capability, Hardware Component, Hardware Inventory, Hardware Type, SACM Interface, Target Endpoint Characterization Record, Target Endpoint Characterization Task, Target Endpoint Classification Task, Target Endpoint Discovery Task, Vulnerability Description Information, Vulnerability Detection Data, Vulnerability Management Capability, Vulnerability Assessment
- Added references to i2nsf definitions in Capability, SACM Component, SACM Interface, SACM Role.
- Added i2nsf Terminology I-D Reference.
- Major Updates to Endpoint, SACM Task, Target Endpoint Identifier.
- Minor Updates to Guidance, SACM Component Discovery, Target Endpoint Label, Target Endpoint Profile.
- Relabeled SACM Task
- Removed Target Endpoint Discovery

Changes from version 10 to version 11:

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

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

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

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

Changes from version 11 to version 12:

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

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

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

Changes from version 12 to version 13:

- Added Virtual Component.

- Major Updates to Capability, Collection Task, Hardware Component, Hardware Type, Security Automation, Subject, Target Endpoint, Target Endpoint Profile.

- Minor Updates to Assertion, Data Plane, Endpoint Characteristics.

Changes from version 13 to version 14:

- Handled a plethora of issues listed in GitHub.

- Pruned some commonly understood terms.

- Narrowing term labels per their definitions.

- In some cases, excised expositional text.

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

Changes from version 14 to version 16:

- moved obsolete definitions into the Appendix (attic).

7. Contributors

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

Email: david.waltermire@nist.gov

Adam W. Montville
Center for Internet Security
31 Tech Valley Drive
East Greenbush, NY 12061
USA

Email: adam.w.montville@gmail.com

David Harrington
Effective Software
50 Harding Rd
Portsmouth, NH 03801
USA

Email: ietfdbh@comcast.net

Brian Ford
Lancope
3650 Brookside Parkway, Suite 500
Alpharetta, GA 30022
USA

Email: bford@lancope.com

Merike Kaeo
Double Shot Security
3518 Fremont Avenue North, Suite 363
Seattle, WA 98103
USA

Email: merike@doubleshotsecurity.com

8. References

8.1. Normative References

[RFC5792] Sangster, P. and K. Narayan, "PA-TNC: A Posture Attribute (PA) Protocol Compatible with Trusted Network Connect (TNC)", RFC 5792, DOI 10.17487/RFC5792, March 2010, <https://www.rfc-editor.org/info/rfc5792>.

[RFC6933] Bierman, A., Romascanu, D., Quittek, J., and M. Chandramouli, "Entity MIB (Version 4)", RFC 6933, DOI 10.17487/RFC6933, May 2013, <https://www.rfc-editor.org/info/rfc6933>.

8.2. Informative References

[I-D.ietf-i2nsf-terminology] Hares, S., Strassner, J., Lopez, D., Xia, L., and H. Birkholz, "Interface to Network Security Functions (I2NSF) Terminology", draft-ietf-i2nsf-terminology-06 (work in progress), July 2018.

[I-D.ietf-netmod-entity] Bierman, A., Bjorklund, M., Dong, J., and D. Romascanu, "A YANG Data Model for Hardware Management", draft-ietf-netmod-entity-08 (work in progress), January 2018.

[I-D.ietf-sacm-vuln-scenario] Coffin, C., Cheikes, B., Schmidt, C., Haynes, D., Fitzgerald-McKay, J., and D. Waltermire, "SACM Vulnerability Assessment Scenario", draft-ietf-sacm-vuln-scenario-02 (work in progress), September 2016.

[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>.

[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>.

[X.1252] "ITU-T X.1252 (04/2010)", n.d..

Appendix A. The Attic

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

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.

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

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

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

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

Authors' Addresses

Henk Birkholz
Fraunhofer SIT
Rheinstrasse 75
Darmstadt 64295
Germany

Email: henk.birkholz@sit.fraunhofer.de

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

Email: jarrett.lu@oracle.com

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

Email: john.sc.strassner@huawei.com

Nancy Cam-Winget
Cisco Systems
3550 Cisco Way
San Jose, CA 95134
USA

Email: ncamwing@cisco.com

Adam Montville
Center for Internet Security
31 Tech Valley Drive
East Greenbush, NY 12061
USA

Email: adam.w.montville@gmail.com