SACM Working Group
Internet-Draft
Intended status: Standards Track
Expires: January 9, 2017

H. Birkholz
Fraunhofer SIT
J. Fitzgerald-McKay
Department of Defense
C. Schmidt
The MITRE Corporation
D. Waltermire
NIST
July 08, 2016

Concise Software Identifiers
draft-birkholz-sacm-coswid-01

Abstract

This document defines a concise representation of ISO 19770-2:2015 Software Identifiers (SWID tags) that is interoperable with the XML schema definition of ISO 19770-2:2015.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

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

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

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

o Remote Attestation, which requires a link between golden (well-known) measurements and software instances [I-D-birkholz-tuda].

SWID tags, as defined in ISO-19770-2:2015 [SWID], provide a standardized format for a record that identifies and describes a specific release of a software product. Different software products, and even different releases of a particular software product, each have a different SWID tag record associated with them. In addition to defining the format of these records, ISO-19770-2:2015 defines requirements concerning the SWID tag lifecycle. Specifically, when a software product is installed on an endpoint, that product's SWID tag is also installed. Likewise, when the product is uninstalled or replaced, the SWID tag is deleted or replaced, as appropriate. As a result, ISO-19770-2:2015 describes a system wherein there is a correspondence between the set of installed software products on an

endpoint, and the presence on that endpoint of the SWID tags corresponding to those products.

SWID tags are meant to be flexible and able to express a broad set of metadata about a software product. Moreover, there are multiple types of SWID tags, each providing different types of information. For example, a "corpus tag" is used to describe an application's installation image on an installation media, while a "patch tag" is meant to describe a patch that modifies some other application. While there are very few required fields in SWID tags, there are many optional fields that support different uses of these different types of tags. While a SWID tag that consisted only of required fields could be a few hundred bytes in size, a tag containing many of the optional fields could be many orders of magnitude larger.

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

Real-world instances of SWID tags can be fairly large, and the communication of SWID tags in use-applications such as those described earlier can cause a large amount of data to be transported. This can be larger than acceptable for constrained devices and networks. Concise SWID tags significantly reduce the amount of data transported as compared to a typical SWID tag. This reduction is enable through the use of CBOR, which maps human-readable labels of that content to more concise integer labels (indices). This allows SWID tags to be part of an enterprise security solution for a wider range of endpoints and environments.

1.1. Requirements notation

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119, BCP 14 [RFC2119].

2. Concise SWID data definition

The following is a CDDL representation of the ISO-19770-2:2015 [SWID] XML schema definition of SWID tags. This representation includes all SWID tag fields and thus supports all SWID tag use cases. The CamelCase notation used in the XML schema definition is changed to hyphen-separated notation (e.g. ResourceCollection is named resource-collection in the Concise SWID data definition). The human-readable names of members are mapped to integer indices via a block of rules at the bottom of the Concise SWID data definition. The 48 character strings of the SWID vocabulary that would have to be stored or transported in full if using the original vocabulary are replaced.

```
concise-software-identity = {
  global-attr,
  * entity-entry,
  * evidence-entry,
  * link-entry,
  * software-meta-entry,
  * payload-entry,
  * any-element-entry,
;  ? content,
  ? corpus,
  ? patch,
  ? media,
  name,
  ? supplemental,
  tag-id,
  ? tag-version,
  ? version,
  ? version-scheme,
}

NMTOKEN = text            ; .regexp to add some validation?
NMTOKENS = [ * NMTOKEN ]

any-attr = text
any-element = any

date-time = time
any-uri = uri

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

meta-type = (
```

```
    global-attr,
    * (text => any-attr),
  )

  meta-element = [
    global-attr,
    * (text => any-attr),
  ]

  resource-collection = (
    global-attr,
    * (directory-entry // file-entry // process-entry // resource-entry)
  )

  file = {
    filesystem-item,
    ? size,
    ? version,
    * (text => any-attr),
  }

  filesystem-item = (
    meta-type,
    ? key,
    ? location,
    name,
    ? root,
  )

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

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

  resource = {
    global-attr,
    type,
  }

  entity = {
    global-attr,
    meta-elements,
```

```
    name,
    ? reg-id,
    role,
    ? thumbprint,
  }

  evidence = {
    global-attr,
    resource-collection,
    ? date,
    ? device-id,
  }

  link = {
    global-attr,
    ? artifact,
    href,
    ? media,
    ? ownership,
    rel,
    ? type,
    ? use,
  }

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

  payload = {
    global-attr,
    resource-collection,
  }
```

```
  tag-id = (0: text)
  name = (1: text)
  ; ambiguous, replaced
  ; content = (
  ;  2: [ * entity / evidence / link / software-meta /
  ;       payload / any-element ]
  ;)
  entity-entry = (2: entity)
  evidence-entry = (3: evidence)
  link-entry = (4: link)
  software-meta-entry = (5: software-meta)
  payload-entry = (6: payload)
  any-element-entry = (7: any-element)
  corpus = (8: bool)
  patch = (9: bool)
  media = (10: text)
  supplemental = (11: bool)
  tag-version = (12: integer)
  version = (13: text)
  version-scheme = (14: NMTOKEN)
  lang = (15: text)
  directory-entry = (16: directory)
  file-entry = (17: file)
  process-entry = (18: process)
  resource-entry = (19: resource)
  size = (20: integer)
  key = (21: bool)
  location = (22: text)
  root = (23: text)
  path-elements = (24: ([ * (directory / file) ]))
  pid = (25: integer)
  type = (26: text)
  meta-elements = (27: ([ * meta-element ]))
  reg-id = (28: any-uri)
  role = (29: NMTOKENS)
  thumbprint = (30: text)
  date = (31: date-time)
  device-id = (32: text)
  artifact = (33: text)
  href = (34: any-uri)
  ownership = (35: ("shared" / "private" / "abandon"))
  rel = (36: NMTOKEN)
  use = (37: ("optional" / "required" / "recommended"))
  activation-status = (38: text)
  channel-type = (39: text)
  colloquial-version = (40: text)
  description = (41: text)
  edition = (42: text)
```

```
  entitlement-data-required = (43: bool)
  entitlement-key = (44: text)
  generator = (45: text)
  persistent-id = (46: text)
  product = (47: text)
  product-family = (48: text)
  revision = (49: text)
  summary = (50: text)
  unspsc-code = (51: text)
  unspsc-version = (52: text)
```

3. Encoding hashes for Concise SWID tags

Concise SWID support hashes that are registered at the Named Information Hash Algorithm Registry and the Hash Function Textual Names Registry. A text string used as a value for hash-alg-id referes to the Hash Function Name in the Hash Function Textual Names table. A number used as a value for hash-alg-id refers the ID in the Named Information Hash Algorithm table.

If a hash value (e.g. a file hash) is to be the content of any-attr or any-element, it MUST be encoded as a coswid-hash array:

```
coswid-hash = [
  hash-alg-id: int / tstr,
  hash-value: bstr,
]
```

Example :

```
file-hash = bstr .cbor coswid-hash
```

4. COSE signatures for Concise SWID tags

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

The ISO-19770-2:2015 XML schema uses XML DSIG to support cryptographic signatures. Concise SWID tags require a different

signature scheme than this. COSE (CBOR Encoded Message Syntax) provides the required mechanism [I-D.ietf-cose-msg]. Concise SWID can be wrapped in a COSE Single Signer Data Object (cose-sign1) that contains a single signature. The following CDDL defines a more restrictive subset of header attributes allowed by COSE tailored to suit the requirements of Concise SWID.

```
signed-software-identity = #6.997(COSE-Sign1-coswid) ; see TBS7 in current COSE
I-D

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

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

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

COSE-Sign1-coswid = [
    protected: bstr .cbor protected-signed-coswid-header,
    unprotected: unprotected-signed-coswid-header,
    payload: bstr .cbor concise-software-identity,
    signature: bstr,
]
```

5. IANA considerations

This document will include requests to IANA:

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

6. Security Considerations

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

A signed SWID tag whose signature is intact can be relied upon to be unchanged since it was signed. If the SWID tag was created by the software author, this generally means that it has undergone no change since the software application with which the tag is associated was installed. By implication, this means that the signed tag reflects the software author's understanding of the details of that software product. This can be useful assurance when the information in the tag needs to be trusted, such as when the tag is being used to convey golden measurements. By contrast, the data contained in unsigned tags cannot be trusted to be unmodified.

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

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

tags and their association with an endpoint reveals information about that endpoint's attack surface.

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

7. Acknowledgements

8. Change Log

Changes from version 00 to version 01:

- o Ambiguity between evidence and payload eliminated by introducing explicit members (while still allowing for "empty" swid tags)
- o Added a relatively restrictive COSE envelope using cose_sign1 to define signed coswids (single signer only, at the moment)
- o Added a defintion how to encode hashes that can be stored in the any-member using existing IANA tables to reference hash-algorithms

First version -00

9. Contributors

10. References

10.1. Normative References

[I-D.ietf-cose-msg]
Schaad, J., "CBOR Object Signing and Encryption (COSE)", draft-ietf-cose-msg-14 (work in progress), June 2016.

[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>.

[RFC7049] Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", RFC 7049, DOI 10.17487/RFC7049, October 2013, <http://www.rfc-editor.org/info/rfc7049>.

[SAM] "Information technology - Software asset management - Part 5: Overview and vocabulary", ISO/IEC 19770-5:2013, November 2013.

[SWID] "Information technology - Software asset management - Part 2: Software identification tag'", ISO/IEC 19770-2:2015, October 2015.

[X.1520] "Recommendation ITU-T X.1520 (2014), Common vulnerabilities and exposures", April 2011.

10.2. Informative References

[I-D-birkholz-tuda]
Fuchs, A., Birkholz, H., McDonald, I., and C. Bormann, "Time-Based Uni-Directional Attestation", draft-birkholz-tuda-01 (work in progress), March 2016.

[I-D.greevenbosch-appsawg-cbor-cddl]
Vigano, C. and H. Birkholz, "CBOR data definition language (CDDL): a notational convention to express CBOR data structures", draft-greevenbosch-appsawg-cbor-cddl-08 (work in progress), March 2016.

Authors' Addresses

Henk Birkholz
Fraunhofer SIT
Rheinstrasse 75
Darmstadt 64295
Germany

Email: henk.birkholz@sit.fraunhofer.de

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

Email: jmfitz2@nsa.gov

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

Email: cmschmidt@mitre.org

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

Email: david.waltermire@nist.gov