The attached DRAFT document (provided here for historical purposes) has been superseded by the following publication:

Publication Number: **NIST Internal Report (NISTIR) 8060**

Title: ***Guidelines for the Creation of Interoperable Software Identification (SWID) Tags***

Publication Date: **April 2016**

- Final Publication: https://doi.org/10.6028/NIST.IR.8060 (direct link: http://nvlpubs.nist.gov/nistpubs/ir/2016/NIST.IR.8060.pdf).
- Information on other NIST Computer Security Division publications and programs can be found at: http://csrc.nist.gov/

DRAFT NISTIR 8060

Guidelines for the Creation of Interoperable Software Identification (SWID) Tags

David Waltermire
Brant A. Cheikes

National Institute of Standards and Technology
U.S. Department of Commerce

DRAFT NISTIR 8060

Guidelines for the Creation of Interoperable Software Identification (SWID) Tags

David Waltermire
Computer Security Division
Information Technology Laboratory

Brant A. Cheikes
Cyber Security Technical Center
The MITRE Corporation

May 2015

U.S. Department of Commerce
Penny Pritzker, Secretary

National Institute of Standards and Technology
Willie May, Under Secretary of Commerce for Standards and Technology and Director

National Institute of Standards and Technology Internal Report 8060
56 pages (May 2015)

Public comment period: *June 1, 2015* through *June 15, 2015*

National Institute of Standards and Technology
Attn: Computer Security Division, Information Technology Laboratory
100 Bureau Drive (Mail Stop 8930) Gaithersburg, MD 20899-8930
Email: nistir8060-comments@nist.gov

Reports on Computer Systems Technology

The Information Technology Laboratory (ITL) at the National Institute of Standards and Technology (NIST) promotes the U.S. economy and public welfare by providing technical leadership for the Nation's measurement and standards infrastructure. ITL develops tests, test methods, reference data, proof of concept implementations, and technical analyses to advance the development and productive use of information technology. ITL's responsibilities include the development of management, administrative, technical, and physical standards and guidelines for the cost-effective security and privacy of other than national security-related information in Federal information systems.

Abstract

This guidance provides an overview of the capabilities and usage of Software Identification (SWID) tags as part of a comprehensive software life cycle. As instantiated in the ISO/IEC 19770-2 standard, SWID tags support numerous applications for software asset management and information security management. This publication introduces SWID tags in an operational context, provides guidance for the creation of interoperable SWID tags, and highlights key usage scenarios for which SWID tags are applicable.

Keywords

software, software asset management, software identification tag, SWID

Acknowledgements

The authors would like to thank Harold Booth of the National Institute of Standards and Technology, and Valery Feldman and Greg Witte of G2, Inc. for their contributions to and review of this report.

Note to Reviewers

This document represents an initial discussion draft of this report. The authors are planning to conduct a number of iterations of this document to further develop the concepts and guidance contained herein based on public feedback. A typical cycle of revision will consist of a two week public comment period followed by a two to three week revision period resulting in an updated discussion draft. The authors plan to conduct three to six iterations of this cycle before finalizing this document. While this is a slight departure from the normal development cycle, the authors believe that this collaborative approach will result in a better set of usable guidance for SWID tag creators.

For this initial draft iteration, review should be primarily focused on the first four sections of this report. Specific attention should be given to the inline questions in these sections. These questions represent areas where a significant degree of feedback is needed to advance this report. Section 5 of this document is being deemphasized since it is less developed than the balance of the document. We have included this section in its current, less-mature state to provide a sense of the desired content of the section. Tightening and clarifying the concepts in this section will be a major focus for the next draft release along with addressing comments received on the rest of the report.

Trademark Information

Document Conventions

This document provides both informative and normative guidance supporting the use of SWID tags. 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 Request for Comment (RFC) 2119. When these words appear in regular case, such as “should” or “may”, they are not intended to be interpreted as RFC 2119 key words.

Some of the requirements and conventions used in this document reference Extensible Markup Language (XML) content. These references come in two forms, inline and indented. An example of an inline reference is: A patch tag is differentiated by the fact that the value of the `@patch` attribute within the `<SoftwareIdentity>` element is “`true`”.

In this example, the notation `<SoftwareIdentity>` can be replaced by the more verbose equivalent “the XML element whose qualified name is SoftwareIdentity”.

The general convention used when describing XML attributes within this document is to reference the attribute as well as its associated element, employing the general form "`@attributeName` for the `<prefix:localName>`". Indented references are intended to represent the form of actual XML content. Indented references represent literal content by the use of a fixed-length font, and parametric (freely replaceable) content by the use of an italic font. Square brackets ‘[]’ are used to designate optional content.

Both inline and indented forms use qualified names to refer to specific XML elements. A qualified name associates a named element with a namespace. The namespace identifies the XML model, and the XML schema is a definition and implementation of that model. A qualified name declares this schema to element association using the format ‘prefix:element-name’. The association of prefix to namespace is defined in the metadata of an XML document and varies from document to document.

Table of Contents

List of Appendices

1 Introduction

ISO/IEC 19770-2 specifies an international standard for *software identification tags*, also referred to as “SWID tags.” A SWID tag is a formatted set of data elements which collectively identify and describe a software product. The first version of the standard was published in 2009, and is designated ISO/IEC 19770-2:2009 [ISO/IEC 19770-2:2009]. A significantly revised version of the standard will be published in 2015, and will be designated ISO/IEC 19770-2:2015. This updated standard is referenced herein as the *SWID specification*. This document provides an overview of the capabilities and usage of the 19770-2:2015 version of SWID tags, focusing on the use of SWID tags as part of comprehensive software asset management (SAM) life cycles and cybersecurity procedures.

Section 1.1 discusses the SAM and cybersecurity problems which motivated the development of SWID tags. Section 1.2 highlights the significant benefits which stakeholders stand to gain as SWID tags become more widely produced and consumed within the marketplace. Section 1.3 describes the purpose and target audiences of this document. Section 1.4 summarizes this section’s key points, and Section 1.5 describes how the rest of this document is organized.

1.1 Problem Statement

Software is part of the critical infrastructure for the modern world. Enterprises as well as individuals routinely acquire software products and deploy them on the physical and/or virtual computing devices they own or operate. ISO/IEC 19770-1, a companion standard to the SWID specification, defines *software asset management* (SAM) as “effective management, control and protection of software assets within an organization.” A core SAM process is *software inventory management*—the process of building and maintaining an accurate and complete inventory of all software products deployed on all of the devices under an organization’s operational control.

Consumers of software products tend to prioritize the features, functions, and usability of software when making purchasing decisions. This often creates incentives for software producers to focus their development practices on these factors. As a result, product *manageability* is often a lesser concern. Reliable and authoritative indicators of SAM life cycle events are often unavailable when products are installed, licensed, patched, upgraded or uninstalled. For this reason there is no consistent, standardized way to automate the processes of *discovering* a software product on a device (i.e., determining which products are present), or *identifying* an installed product by collecting key descriptive characteristics such as its exact version, license keys, patch level, associated files in device storage areas, etc. Instead, software products are installed in idiosyncratic ways that may differ substantially by product provider, operating environment, and device. This creates management challenges for enterprise IT managers who need to track software installed within their heterogeneous networked environments.

Accurate software inventories of enterprise managed devices are needed to support higher-level business and cybersecurity functions. For example:

- **Chief Information Officers (CIOs):** To ensure compliance with software license agreements, CIOs need to know how many copies of a given product are installed. To ensure they are not paying for unneeded licenses, CIOs need to know where specific copies are installed and whether they are in active use.

- **Chief Information Security Officers (CISOs):** CISOs and operations personnel need accurate and complete software inventories to ensure that all deployed software assets are authorized, appropriately patched, free of known exploitable weaknesses, and configured in ways consistent with their organizations' security policies.

To address these needs, commercial products are offered that provide software inventory and discovery capabilities. These products employ a variety of proprietary techniques to discover and identify installed software applications. These techniques vary greatly in their accuracy, coverage of operating environments, identification of specific installed software, quality of reports produced, and amount of descriptive detail they are able to provide about each discovered application. As a result, different inventory and discovery products often reach different conclusions when inventorying the same device. For enterprises which employ inventory and discovery tools from multiple vendors, variations in report content can make it difficult or impossible to correlate findings across those tools. Finally, proprietary solutions often do not interoperate with other products, making it difficult and expensive to integrate a new inventory or discovery product into an existing infrastructure.

One way to solve this problem is for software providers to adopt standard methods whereby routine inventory and discovery procedures leave indicators behind with enough consistency, detail, and fidelity to support all required SAM and cybersecurity objectives. The SWID tag standard has been developed to provide a data format for such indicators.

1.2 SWID Tag Benefits

SWID tags offer benefits to both creators of software products and those who acquire and use those software products. The SWID specification identifies these stakeholders as:

Tag producers: Organizations and entities that create SWID tags for use by others in the market. Ideally, the organizations involved in creating, licensing, and/or distributing software products will also create the tags which accompany their products. This is ideal because these organizations are best able to ensure that the tags contain correct and complete data. In other cases tags may be produced and distributed by other entities, including third parties and even automated tools.

Tag consumers: Organizations and entities that use information contained in SWID tags associated with deployed software products to support higher-level, software-related business and cybersecurity functions. Categories of tag consumers include software consumers, inventory/discovery tools, and inventory-based cybersecurity tool providers (e.g., providers of software vulnerability management products, which rely on accurate inventory information to support accurate vulnerability assessment), and organizations that use these tools.

The implementation of SWID tags beneficially supports these stakeholders throughout the entire software lifecycle—from software creation and release through software installation, management, and de-installation. As more software creators also become tag producers by releasing their products with SWID tags, more consumers of software products become able to also consume the associated tags. This promotes a "virtuous cycle" where all stakeholders gain a variety of benefits including:

- The ability to consistently and accurately identify software products that need to be managed for any purpose, such as for inventory, licensing, cybersecurity, or for the management of software and software dependencies.
- The ability to exchange software information between software producers and consumers in a standardized format regardless of software creator, platform, or management tool.
- The ability to identify and manage software products equally well at any level of abstraction, regardless of whether a product consists of a single application, or one or more groups or bundles.
- The ability to correlate information about installed software with other information including list(s) of authorized software, related patches, configuration settings, security policies, and advisories.
- The ability to automatically track and manage software license compliance and usage, by combining information within a SWID tag with independently-collected software entitlement data.
- The ability to record details about the deployed footprint of installed products on devices, such as the list of supporting software components, executable and data files, system processes, and generic resources that may be included in the installation (e.g., device drivers, registry settings, user accounts).
- The ability to identify all organizational entities associated with the installation, licensing, maintenance, and management of a software product on an on-going basis, including software creators, software licensors, packagers, distributors external to the software consumer, as well as various entities within the software consumer.
- Through the optional use of digital signatures, the ability to validate that information within the tag comes from a known source and has not been corrupted.

1.3 Purpose and Audience

This document has three purposes. First, it provides a high-level description of SWID tags, in order to increase familiarity with the standard. Second, it provides guidance on the creation of specific types of SWID tags that supplements the SWID tag specification. Lastly, it presents a set of operational usage scenarios together with guidelines to be followed by tag creators when preparing tags (i.e., populating the data elements that comprise tags) for use in those scenarios. By following these guidelines, tag creators can have confidence they are providing all the necessary data, with the requisite data quality, needed to achieve the operational goals of each tag usage scenario.

The material herein addresses three distinct audiences. The first audience is *software providers*, the individuals and organizations that develop, license, and/or distribute commercial, open source, and custom software products. Software providers also include organizations that develop software solely for in-house use. This document will help providers understand the problems addressed by SWID tags, why providers' participation is essential to solving those problems, and how providers may produce and distribute tags which meet the needs of a wide range of usage scenarios.

The second audience is *providers of inventory-based products and services*, the individuals and organizations that develop tools for discovering and managing software assets for any reason, including to secure enterprise networks using information from standard inventory processes. This audience has unique needs due to the fact that their products and services will consume and utilize information in SWID tags as tags increasingly become available on endpoints. For inventory-based product providers, this document describes usage scenarios where the presence of properly implemented SWID tags materially enhances the quality and coverage of information which their products may collect and utilize about installed software products. By offering guidance to *software providers* on how to properly implement tags to support these usage scenarios, this document helps *inventory-based product providers* (and providers of other related IT management tools) prepare their specialized products to take full advantage of those tags when available.

The third audience is *software consumers*, the individuals and organizations that install and use commercial, open source, and/or in-house developed software products. This document helps software consumers understand the benefits of software products which are delivered with SWID tags, and why they should encourage software providers to deliver products with SWID tags that meet all the requirements of consumers' anticipated usage scenarios.

This document seeks to help each of the three audiences understand how their respective goals are interrelated. Consumers are on the front lines, trying to cope with software management and cybersecurity challenges that require accurate software inventory. They want to address these challenges in a way that promotes a low total cost of ownership for the software they manage. Consumers need to understand how SWID tags can help them, need providers to supply high-quality tags, and need implementers of inventory-based tools to collect and utilize tags. Providers need to recognize that adding tags to their products will make their products more useful and more manageable, and also need this recognition to be reinforced by clear consumer demand signals. Inventory-based tool implementers are uniquely positioned to recognize how tags could make their products more reliable and effective, and could work constructively with both consumers and providers to promote software tagging practices.

1.4 Section Summary

These are the key points of this section:

- ISO/IEC 19770-2 specifies an international standard data format for software identification (SWID) tags. The first version of the standard was published in 2009 (designated 19770-2:2009) and a significantly revised version will be published in 2015 (designated 19770-2:2015). This document pertains to SWID tags as specified in 19770-2:2015.

- SWID tags were developed to help enterprises meet pressing needs for accurate and complete software inventories to support higher-level business and cybersecurity functions.

- Tags provide an array of benefits to organizational entities which create tags as well as to those which consume tags.

- Three audiences have interrelated goals related to SWID tags and tagging practices:

  - *Software consumers* are trying to cope with the challenges of conducting an accurate software inventory and the associated cybersecurity issues. They need software providers to supplying tags along with their products as a common practice.

  - *Software providers* need to increase the manageability of their products for their customers. To invest the resources necessary to become tag providers, they need consumers to send clear signals that they value product manageability as much as features, functions, and usability.

  - *Inventory-based tool providers* need to commit to SWID tags as their primary method for identifying software, and at the same time need more tags to become available to make their specialized tools more reliable and effective. They act as software providers as well as software consumers, and thus have the needs and goals of both audiences.

- This document seeks to raise awareness of the SWID tag standard, promote understanding of the business and cybersecurity benefits which may be obtained through increased adoption of tag standards and practices, and provide detailed guidance to both producers and consumers of SWID tags.

1.5 Document Structure

The remainder of this document is organized into the following sections and appendices:

- Section 2 presents a high-level overview of the SWID tag standard. This section will be of interest to all audiences, as it explains what a SWID tag is, and how tags encode a variety of identifying and descriptive data elements about software products.
- Section 3 provides implementation guidance that addresses issues common to all situations in which tags are deployed and processed on information systems. The intent of this guidance is to be broadly applicable to common IT usage scenarios that are relevant to both public and private sector organizations.
- Section 4 provides implementation guidance that varies according to the type of tag being implemented.
- Section 5 describes several usage scenarios for software asset management and for software integrity management. These are not intended to represent an exhaustive or conclusive list of possible SWID applications, but provide informative examples regarding the use of the SWID specification to accomplish various organizational needs.
- Appendix A presents a list of selected acronyms used in this document.
- Appendix B provides the references for the document.

2 SWID Tag Overview

A SWID tag is a standard format for a set of data elements that identify and describe a software product. SWID tags are formatted as XML documents. When a software product is installed on a computing device, SWID tags for that product should also be installed or otherwise become discoverable on that device. When a product is uninstalled from a device, all associated tags should be removed.[1] In this way, the presence of tags on a device serves as evidence of the presence of the related software products on that device described by the tags. The SWID tag specification defines these behaviors, as well as related behaviors associated with software licensing, patching, and upgrading

Because software products and their tags are logically separate entities, it is important to maintain clear distinctions between SWID tags and both (a) the products that are identified and described by SWID tags, and (b) the entities and processes involved in SWID tag creation, deployment, storage, and retrieval. This document uses the term *tagged software product* (or, simply, *tagged product*) to refer to situations where a product is installed on a device, and one or more tags for that product are discoverable (whether stored explicitly or obtainable through an interface) on the device. Saying that a product is tagged does not necessarily mean that all its associated tags are created by the product provider; in fact, as this section will make clear, the various types of tags that may be associated with a given product may be supplied by a variety of organizational entities and automated tools.

This section presents a high-level description of SWID tag data elements as specified in the SWID specification. The material presented here is sufficient for most audiences to acquire a general understanding how SWID tags may be used to identify and describe software products. To correctly implement tags, interested readers may want to obtain the ISO specification and the corresponding XML schema definition (XSD). The XSD for SWID tags conformant with the 2015 specification may be downloaded from:

http://standards.iso.org/iso/19770/-2/2015/schema.xsd

The remainder of this section is organized as follows. Section 2.1 discusses expectations regarding where SWID tags reside relative to the products they identify, and how the location of a tag may or may not relate to the computing device(s) where the tagged product may be executed. Section 2.2 describes four types of tags defined in the specification. Section 2.3 discusses the various scenarios by which a SWID tag is made available on a device. Section 2.4 presents an overview of the basic data elements that comprise a SWID tag. Section 2.5 discusses how SWID tags may be authenticated. Section 2.6 presents examples of the four tag types, and Section 2.7 concludes with a summary of key points from this section.

[1] On devices that have file systems, the SWID tag for an installed software product should be discoverable in a directory labeled "swidtag" that is either at the same level as the product's installation directory, or is an immediate sub-directory of the product's installation directory. Alternatively, or on devices without file systems, tags should be accessible through platform-specific interfaces and/or maintained in platform-specific storage locations.

2.1 Scope Note

As the Information Technology market has evolved, the concept of an "installed software product" has become increasingly complicated. The simplest concept of an "installed software product" is software that is able to be loaded into memory and executed on a computing device by virtue of being *physically stored* on that device. Software is "physically stored" on a computing device if it is recorded in a persistent storage component that is itself part of the hardware comprising the computing device.[2] This document is primarily concerned with the use of SWID tags to identify software products and discover *where they are stored*, because it is generally assumed that where a product is stored also determines where (and often by whom) that product may be executed.

The assumption that software products are physically stored on the same computing devices used to execute them is often wrong. For example, through the use of high-performance networking technologies, a software product can be physically stored on a network-attached storage (NAS) device, then executed seamlessly on any computing device able to access that NAS device. In situations like these, products and their tags should co-reside on the NAS device, and inventory tools should consider the products part of the inventory of the NAS device, not part of the inventory of each accessible computing device. In other words, storage location matters more than (and determines tag placement more than) where a product may be executed.

As another example, consider removable media devices such as high-capacity USB thumb drives and SD memory cards. Once a software product is installed on such removable media, it may become executable on a computing device immediately upon insertion of the media. In this scenario, the product should be considered part of the inventory of the removable media, not part of the inventory of whichever computing device it happens to be plugged into, and the product tag should reside along with the product on the removable media.

The rise of virtualization technology further clouds the issue, as it changes the very definition of what it means to be a computing device, and introduces the prospect of virtual devices being created, inventoried, and then destroyed all in the space of mere moments. When software products are installed on a virtual machine that is powered down, inactive, and stored somewhere as a machine image, those products should not be considered to exist in inventory. Consequently it does not make sense for the associated product tags to be stored or discoverable separately from the virtual machine image. But when the virtual machine is activated, loaded into memory on a physical device and assigned to a hypervisor, it should behave as if it were a real device; the tags for all products installed on the virtual machine should reside within the virtual environment so they can be associated with the virtual machine. In this scenario, tags are considered to be physically stored in virtual machine space rather than physical machine space.

Finally, computing innovations such as "software as a service" and "containerization" are challenging the basic notion of what a "software product" fundamentally is. These concepts that rely on the use of ephemeral code create a natural tension between the locality of installation and

---

[2] Software present on removable media (e.g., a USB thumb drive or SD memory card) that is plugged into a computing device is considered physically stored on the computing device according to this definition.

the locality of use. When a software application is operated remotely as a service, it should be considered to be installed on the remote server rather than on the client device. But when a product is containerized and delivered to a client device for execution, that product becomes part of the client device's product inventory, however transiently.

In summary, the general rule for SWID tag placement is that tags should reside on the same storage device that holds the tagged product. Although tag consumers often may infer that a product is executable on the same device where it is stored, they must take care to distinguish cases where products may be executable on devices elsewhere within the enterprise.

2.2 Tag Types

The SWID specification defines four types of SWID tag: *primary*, *supplemental*, *patch*, and *corpus*. With rare exceptions, once a tag of any of these types is installed on a device, it should never be modified, only replaced or removed entirely.[3] The intended use of each tag type is described in the subsections below.

2.2.1 Primary Tags

Each tagged product must provide a single tag which, at a minimum, will furnish values for all data elements that are designated "mandatory" in the SWID specification. This is referred to as the product's *primary* tag. A minimal primary tag supplies the name of the product (as a string), a globally unique identifier for the tag, and basic information identifying the tag's creator. It is important to note that the creator of a tag might not be the software provider. This distinction is discussed in section 2.4.2.

A globally unique tag identifier is essential information in many usage scenarios because it may be used as a globally unique *proxy identifier* for the tagged product. The tag identifier can be considered a proxy identifier because there is a one-to-one binding between the tag and the software it identifies. For example, in some contexts it will be more efficient in terms of data transmission and processing costs for inventory and discovery tools to identify and report tagged products using only their tag identifiers, rather than their fully populated tags.

Ideally, the product vendor is also the creator of that product's primary tag; however, the standard permits other parties (including automated tools) to create tags for products in cases where product vendors have declined to do so or have delegated this responsibility to another party.

2.2.2 Supplemental Tags

Because a minimally-populated primary tag is unlikely to furnish data values sufficient for all

---

[3] It cannot be assumed that inventory tools will fully and routinely check for changes related to previously-discovered tags. The preferred method for *correcting tag errors* is to replace an incorrect tag with a correct tag. When correcting a tag in this way, the new tag's `@tagVersion` (cf. Section 2.4.1) is set to a larger value. The preferred method for *adding information* about a product (e.g., installation timestamps, license keys, etc.) is to install a supplemental tag (cf. Section 2.2.2) linked to the product's primary tag (cf. Section 2.2.1). In either case, new tags should be used to avoid invalidating the XML digital signature of the original tag.

usage scenarios of interest, the standard allows for any number of *supplemental* tags to be installed, either at the same time the primary tag is installed or at any time thereafter. Supplemental tags may, but need not, be created by the same entity that created the primary tag. Thus supplemental tags may be used by automated tools to augment a primary tag with additional site-specific information, such as license keys, contact information for local responsible parties, etc.

Each supplemental tag contains a pointer to the product's primary tag (cf. Section 2.4.4 on the `<Link>` element). When supplemental tags are present, a tag consumer may create a complete record of the information describing a product by combining the data elements in the product's primary tag with the data elements in any linked supplemental tags.

A supplemental tag is intended to furnish data values which augment and do not conflict with data values provided by the primary tag and any other of the product's supplemental tags. If conflicts are detected, data in the primary tag should be considered the most reliable, and tools should report all other conflicting data as exceptions. For example, the mandatory product name recorded in a supplemental tag should match the product name recorded in the product's primary tag, but if they are different, the name recorded in the primary tag should be considered the most reliable name.

2.2.3 Patch Tags

The SWID specification defines a *patch* as "a software component that, when installed, directly modifies files or device settings related to a different software component without changing the version number or release details for the related software component." Patches are commonly used to efficiently repair defects in software products with large and complex codebases, such as operating systems and major applications.

When a tagged product is patched, a patch tag should be installed as part of the patch procedure. It is also expected that if a patch is uninstalled, the associated patch tag should be removed. A patch tag is a special kind of primary tag: it records the installation of a "product" (i.e., the patch) which may have a name, version, etc., distinct from the patched product, but includes information linking it with the primary tag of the product to which the patch was applied (cf. Section 2.4.4 on the `<Link>` element). In this way patch tags may be used to determine whether an installed product has all required patches installed.

A patch will likely also include a manifest of the patched files (cf. Section 2.4.6 on the `<Payload>` element) which can be used to verify that the actual patched files are present on the device. This allows for confirmation that the patch has been correctly installed, preventing a malicious actor from deploying a patch tag that misrepresents the installation status of a patch.

In contrast with a patch, an *upgrade* is a more complete release for a product's codebase that also changes the product's version number and/or release details. When this occurs, all tags associated with the original (pre-upgrade) product should be removed, and new tags installed.

Unlike supplemental tags, which are used to augment the identifying and descriptive data elements that are furnished in a product's primary tag, patch tags describe localized changes made to a product's codebase. Such localized changes may be named, versioned, and tracked separately from the base product. Thus the identifying and descriptive data elements contained in a patch tag should be treated as identifying and describing the patch rather than the product to which the patch is applied; for example, the product name and version recorded in a patch tag need not match the product name and version recorded in the product's primary tag, and may instead be used to record the name and version of the patch as assigned by the product provider.

2.2.4 Corpus Tags

When products and patches are distributed to a device in preparation for installation, they typically are deployed in a "pre-installation" structure, often called a *software installation package*. This pre-installation structure may be stored in a file, on removable media, or on a network storage device. While primary, supplemental, and patch tags are used to identify and describe installed software products, they do not identify and describe a software installation package that can be used to install a software product. The availability of software identification and descriptive information for a software installation package enables verification of the software package and authentication of the organization releasing a package. The SWID specification defines *corpus* tags for vendors and distributors to use to identify and describe products in such a pre-installation state.

Corpus tags may be used by consumers to verify the integrity of an installable product and authenticate the issuer of the installation before carrying out the installation procedure. If a manifest of the installation files is included in the corpus tag (cf. Section 2.4.6 on the `<Payload>` element), installation package tampering can be detected prior to installation. When combined with other licensing data, corpus tags also may aid consumers in confirming whether they have a valid license for a product before they install it.

Corpus tags are, in essence, pre-installation primary tags. In most respects, the identifying and descriptive data elements furnished in a corpus tag (e.g., product name, version, etc.) should be the same as the data elements that will be contained in the product's primary tag post-installation. Due to the fact that software products are typically packaged or "containerized" in special pre-installation formats, the Payload portion (cf. Section 2.4.6) of a corpus tag will likely differ from the Payload portion of the primary tag that is eventually deployed on devices post-installation.

2.3 Tag Deployment

A tag may be created:

- During a product's build/release process by an authoritative source,
- During an endpoint-scanning process by a non-authoritative source (e.g. by an automated software discovery tool), or
- As the result of a post-release analytic processes, by a non-authoritative source which obtains a copy of a product after its release to market, that then uses reverse-engineering and analysis techniques to create a tag.

Once a tag is created, it may be deployed in three main ways. Tag deployment makes a tag discoverable by tag consumers. The preferred method of tag deployment is for a tag to be incorporated into the product's installation package, and then installed on an endpoint as part of the software installation procedure. This method requires that the tag creator who creates the tag is in a position to ensure that it is included in the installation package.

A second method of tag deployment is to store them in publicly accessible repositories. Doing so provides significant value to software consumers because it enables them to:

- Confirm that a tag that has been discovered on an endpoint has not been modified,
- To restore a tag which has been inadvertently deleted,
- To correct a tag which has been improperly modified, and
- To utilize the information in the tag to support various software-related management and analysis processes.

A third method of tag deployment is implicit. Some operating environments furnish native package management systems which, when properly used to install products within those environments, automatically record all the information required to populate required data elements in a tag. In these situations, software installation systems may avoid explicit preparation and deployment of a tag on a system, as long as the native package manager provides a published interface allowing valid tags to be obtained. When a tag is produced on the installation host in this way, it will not be possible to verify the integrity of the tag produced unless an equivalent tag is also produced using the second method described above.

2.4 Basic Tag Elements

This section discusses the basic data elements of a SWID tag. This discussion will also explain how the four tag types described above are distinguished from each other.

A SWID tag (whether primary, supplemental, patch, or corpus) is represented as an XML root element with several sub-elements. `<SoftwareIdentity>` is the root element, and is described in Section 2.4.1. The following sub-elements are used to express distinct categories of product information: `<Entity>` (Section 2.4.2), `<Evidence>` (Section 2.4.3), `<Link>` (Section 2.4.4), `<Meta>` (Section 2.4.5), and `<Payload>` (Section 2.4.6). Section 2.5 briefly discusses how digital signatures within SWID tags may be used to verify a tag's integrity and to authenticate the signer of a tag.

2.4.1 <SoftwareIdentity>: The Root of a SWID Tag

Besides serving as the container for all the sub-elements described in later subsections, the `<SoftwareIdentity>` element provides attributes to record the following descriptive properties of a software product:

- `@name`: the string name of the software product or component as it would normally be referenced, e.g., "ACME Roadrunner Management Suite". A value for `@name` is **required**.

- `@version`: the detailed version of the product, e.g., "4.1.5". A value for `@version` is **optional** and defaults to "0.0".

- `@versionScheme`: a label describing how version information is encoded, e.g., "`multipartnumeric`". A value for `@versionScheme` is **optional** and defaults to "`multipartnumeric`".

- `@tagId`: a globally-unique identifier that may be used as a proxy identifier in other contexts to refer to the tagged product. A value for `@tagId` is **required**.

- `@tagVersion`: an integer which allows one tag for a software product to supersede another, without indicating any change to the underlying software product being described. This value can be increased to correct errors in or to add new information to an earlier tag. A value for `@tagVersion` is **optional** and defaults to 0 (zero).

  It should be considered an error if multiple tags are found for the same installed product with the same the same `@tagId` and a different `@tagVersion`. If this occurs, the tag with the highest `@tagVersion` should be used.

- `@supplemental`: a boolean value which, if set to `true`, indicates that the tag type is *supplemental*, and if set to `false`, indicates that the tag type is *primary*. A value for `@supplemental` is **optional** and defaults to `false`.

- `@patch`: a boolean value which, if set to `true`, indicates that the tag type is *patch*. A value for `patch` is **optional** and defaults to `false`. (Note: if `@patch` is set to `true`, `@supplemental` must be `false`.)

- `@corpus`: a boolean value which, if set to `true`, indicates that the tag type is *corpus*. A value for `@corpus` is **optional** and defaults to `false`.

2.4.1.1 Example 1—Primary Product Tag

This example illustrates a primary tag for version 4.1.5 of a product named "ACME Roadrunner Management Suite Coyote Edition." The globally unique tag identifier, or `@tagId`, is "com.acme.rms-ce-v4-1-5-0". The `<Entity>` element (cf. Section 2.4.2) is included so the example illustrates all data values required in a minimal tag that conforms to the ISO standard. Any additional identifying data (not shown) would appear in place of the ellipsis.

```
<SoftwareIdentity
  xmlns="http://standards.iso.org/iso/19770/-2/2015/schema.xsd"
  name="ACME Roadrunner Management Suite Coyote Edition"
  tagId="com.acme.rms-ce-v4-1-5-0"
  tagVersion="0"
  version="4.1.5">
  <Entity
    name="The ACME Corporation"
```

```
    regid="acme.com"
    role="tagCreator softwareCreator"/>
  …
</SoftwareIdentity>
```

2.4.1.2 Example 2—Supplemental Tag

This example illustrates a supplemental tag for an already installed product. The globally unique identifier of the supplemental tag “com.acme.rms-sensor-1”. The `<Entity>` element (cf. Section 2.4.2) is included so the example illustrates all data values required in a minimal tag that conforms to the standard. The `<Link>` element (cf. Section 2.4.4) is included to illustrate how a supplemental tag may be associated with the primary tag shown above in Section 2.4.1.1. This supplemental tag may be supplying additional installation details which are not included in the product’s primary tag (e.g., site-specific information such as contact information for the local product steward). These details would appear in place of the ellipsis.

```
<SoftwareIdentity
  xmlns=http://standards.iso.org/iso/19770/-2/2015/schema.xsd
  name="ACME Roadrunner Management Suite Coyote Edition"
  tagId="com.acme.rms-sensor-1"
  supplemental="true">
  <Entity
    name="The ACME Corporation"
    regid="acme.com"
    role="tagCreator softwareCreator"/>
  <Link
    rel="related"
    href="swid:com.acme.rms-ce-v4-1-5-0">
  …
</SoftwareIdentity>
```

2.4.1.3 Example 3—Patch Tag

This example illustrates a patch tag for a previously installed product. The name of the patch is “ACME Roadrunner Service Pack 1”, and its globally unique tag identifier is “com.acme.rms-ce-sp1-v1-0-0”. `<Entity>` and `<Link>` elements are illustrated as before. Any additional identifying data (not shown) would appear in place of the ellipsis.

```
<SoftwareIdentity
  xmlns="http://standards.iso.org/iso/19770/-2/2015/schema.xsd"
  name="ACME Roadrunner Service Pack 1"
  tagId="com.acme.rms-ce-sp1-v1-0-0"
  patch="true"
  version="1.0.0">
  <Entity
    name="The ACME Corporation"
    regid="acme.com"
```

```
      role="tagCreator softwareCreator"/>
    <Link
      rel="patches"
      href="swid:com.acme.rms-ce-v4-1-5-0">
  …
</SoftwareIdentity>
```

2.4.2 <SoftwareIdentity> Sub-Element: <Entity>

Every SWID tag must identify, at minimum, the organizational or individual entity which created the tag. Entities having other roles associated with the identified software product, such as its creator, licensor(s), distributor(s), etc., may optionally be identified. These entities are identified using `<Entity>` elements contained within the `<SoftwareIdentity>` element. Each `<Entity>` element provides the following attributes:

- `@name`: the string name of the entity, e.g., "The ACME Corporation". A value for `@name` is **required.**

- `@regid`: the "registration identifier" of the entity (further discussed below). A value for `@regid` is **required** when the Entity element is used to identify the tag creator (i.e., @role="tagCreator"), otherwise `@regid` is **optional** and defaults to "invalid.unavailable".

- `@role`: the role of the entity with respect to the tag and/or the product identified by the tag. Every `<Entity>` element must contain a value for `@role`, and additionally, every tag must contain an `<Entity>` element identifying the tag creator. Values for `@role` are selected from an extensible set of allowed tokens, including these:

    - **`aggregator:`** entities which package sets of products and make them available as single installable items

    - **`distributor:`** entities which handle distribution of products developed by others

    - **`licensor:`** entities which handle licensing on behalf of others

    - **`softwareCreator:`** entities which develop software products

    - **`tagCreator:`** entities which create SWID tags

Values for `@regid` must be URI references as described in RFC 3986 [RFC 3986]. To ensure interoperability and allow for open source project support, the specification recommends in section 6.1.5.2 that tag creators do the following when creating a value for `@regid`:

- Unless otherwise required, the URI should utilize the `http` scheme.

- If the http scheme is used, the "`http://`" may be left off the regid string.
- Unless otherwise required, the URI should use an absolute-URI that includes an authority part, such as a domain name.
- To ensure consistency, the absolute-URI should use the minimum string required (for example, `example.com` should be used instead of `www.example.com`.

The example below illustrates a SWID tag containing two `<Entity>` elements. The first `<Entity>` element identifies the single organization which is both the software creator and the tag creator, and a second element identifies the organization which is the software's distributor:

```
<SoftwareIdentity …>
  …
  <Entity
    name="The ACME Corporation"
    regid="acme.com"
    role="tagCreator softwareCreator"/>
  <Entity
    name="Coyote Services, Inc."
    regid="mycoyote.com"
    role="distributor"/>
  …
</SoftwareIdentity>
```

2.4.3 <SoftwareIdentity> Sub-Element: <Evidence>

Not every software product installed on a device will be supplied with a tag. When a tag is not found for an installed product, third-party software inventory and discovery tools will continue to be used to discover untagged products residing on devices. In these situations, the inventory or discovery tool may generate a primary tag on-the-fly to record the newly-discovered product. The optional `<Evidence>` element may then be used to store results from the scan that explain why the product is believed to be installed. To that end, the `<Evidence>` element provides two attributes and four sub-elements, all of which are optional:

- `@date`: the date the evidence was collected.
- `@deviceId`: the identifier of the device from which the evidence was collected.
- `<Directory>`: filesystem root and directory information for discovered files.
- `<File>`: files discovered and believed to be part of the product.
- `<Process>`: related processes discovered on the device.
- `<Resource>`: other general information which may be included as part of the product.

Note that `<Evidence>` is represented in a SWID tag in the same manner as `<Payload>` (cf.

Section 2.4.6). There is a key difference, however, between <Evidence> and <Payload> data. The <Evidence> element is used by discovery tools that identify untagged software. Here the discovery tool creates a SWID tag based on data discovered on a device. In this case, the <Evidence> element indicates only what was discovered on the device, but this data cannot be used to determine whether discovered files match what a software provider originally released or what was originally installed. In contrast, <Payload> data supplies information from an authoritative source (typically the software provider or a delegate), and thus may be used, for example, to determine if files in a directory match the files that were designated as being installed with a software component or software product.

The example below illustrates a SWID tag containing an <Evidence> element. The evidence consists of two files discovered in a folder named "rrdetector" within the device's standard program data area:

```
<SoftwareIdentity …>
  …
  <Evidence date="11-28-2014" deviceId="mm123-pc.acme.com">
    <Directory root="%programdata%" location="rrdetector">
      <File name="rrdetector.exe" size="532712"/>
      <File name="sensors.dll" size="13295"/>
    </Directory>
  </Evidence>
  …
</SoftwareIdentity>
```

2.4.4 <SoftwareIdentity> Sub-Element: <Link>

Modeled on the HTML [LINK] element, <Link> elements are used to record a variety of relationships between a SWID tag and other items. One typical use of a <Link> element is to associate a supplemental or patch tag to a primary tag. Other uses include pointing to standard licenses, vendor support pages, and installation media. The <Link> element has two required attributes:

- @href: the value is a URI pointing to the item to be referenced.
- @rel: the value specifies the type of relationship between the SWID tag and the item referenced by @href.

A number of additional optional attributes, which are not discussed in this section, support specialized situations.

The example below illustrates how a <Link> element may be used to associate a patch tag with the tag for the patched product:

```
<SoftwareIdentity
  …
  name="ACME Roadrunner Service Pack 1"
```

```
  tagId="com.acme.rms-ce-sp1-v1-0-0"
  patch="true"
  version="1.0.0">
  …
  <Link
    rel="related"
    href="swid:com.acme.rms-ce-v4-1-5-0">
  …
</SoftwareIdentity>
```

In this example, the patch has its own `@tagId` and `@version`, and links to the patched product tag using that product's `@tagId`.

2.4.5 <SoftwareIdentity> Sub-Element: <Meta>

Meta elements are used to record an array of optional metadata attributes related to the tag or to the product. Several `<Meta>` attributes of interest are highlighted below:

- `@activationStatus`: identifies the activation status of the product with respect to any licensing arrangements, e.g., `Trial`, `Serialized`, `Licensed`, `Unlicensed`, etc.

- `@colloquialVersion`: the informal version of the product (i.e., `2013`). The colloquial version may be the same through multiple releases of a software product where the version specified in `<SoftwareIdentity>` is much more specific and will change for each software release.

- @edition: the variation of the product, e.g., Home, Enterprise, Professional, Standard, Student.

- `@product`: the base name of the product, exclusive of vendor, colloquial version, edition, etc.

- `@revision`: the informal or colloquial representation of the sub-version of the product (e.g. SP1, R2, RC1, Beta 2, etc.). Whereas the `<SoftwareIdentity>` element's `@version` attribute will provide exact version details, the `@revision` attribute is intended for use in environments where reporting on the informal or colloquial representation of the software is important, for example, if for a certain business process an organization decides that it must have Service Pack 1 or later of a specific product installed on all devices, they can use the revision data value to quickly identify any devices that do not meet this requirement.

In the example below, a `<Meta>` element is used to record the fact that the product is installed on a trial basis, and to break out the full product name into its component parts:

```
<SoftwareIdentity …>
  …
```

```
  name="ACME Roadrunner Detector 2013 Coyote Edition SP1"
  tagId="com.acme.rd2013-ce-sp1-v4-1-5-0"
  version="4.1.5">
  …
  <Meta
    activationStatus="trial"
    product="Roadrunner Detector"
    colloquialVersion="2013"
    edition="coyote"
    revision="sp1"/>
  …
</SoftwareIdentity>
```

2.4.6 <SoftwareIdentity> Sub-Element: <Payload>

The optional `<Payload>` element is used to enumerate the items (files, folders, license keys, etc.) which may be installed on a device when a software product is installed. In general, `<Payload>` is used to indicate the files that may be installed with a software product and will often be a superset of those files (i.e., if a particular optional component is not installed, the files associated with that component may be included in the `<Payload>`, but not installed on the device.)

The `<Payload>` element is a container for `<Directory>`, `<File>`, `<Process>`, and/or `<Resource>` elements, similar to the <Evidence> element. This example illustrates a primary tag with a `<Payload>` describing two files in a single directory:

```
<SoftwareIdentity …>
  …
  <Payload>
    <Directory root="%programdata%" location="rrdetector">
      <File name="rrdetector.exe" size="532712"/>
      <File name="sensors.dll" size="13295"/>
    </Directory>
  </Payload>
  …
</SoftwareIdentity>
```

2.5 Authenticating SWID Tags

Because SWID tags are documents discoverable on a device, they are vulnerable to unauthorized or inadvertent modification like any other document. To identify tag modifications, it is necessary to validate that a SWID tag collected during an inventory or discovery process has not had specific elements of the tag altered. Digital signatures embedded within a SWID tag can be used to validate that changes have not been made and to prove the authenticity of the tag signer.

Section 6.1.10 of the SWID specification states that:

> Digital signatures are not a mandatory part of the SWID tag standard, and can be used as

> required by any tag producer to ensure that sections of a tag are not modified, and/or to provide authentication of the signer. If signatures are included in the software identification tag, they shall follow the W3C recommendation defining the XML signature syntax which provides message integrity authentication as well as signer authentication services for data of any type.

This text is referencing the W3c note on *XML Advanced Electronic Signatures (XAdES)* [XAdES] which defines a base signature form and six additional signature forms.

Digital signatures use the `<Signature>` element as described in the W3C XML Signature Syntax and Processing (Second Edition) specification [xmldsig-core] and the associated schema.[4] Users may also include a hexadecimal hash string (the "thumbprint") to document the relationship between the tag entity and the signature, using the `<Entity>` `@thumbprint` attribute.

Section 6.1.10 of the ISO specification references the XAdES with Time-Stamp (XAdES-T) form stating that:

> When a signature is utilized for a SWID tag, the signature shall be an enveloped signature and the digital signature must include a timestamp provided by a trusted timestamp server. This timestamp must be provided using the XAdES-T form.
>
> The SWID tag must also include the public signature for the signing entity.

The SWID tag specification in section 6.1.10 also requires that a digitally-signed SWID tag enable tag consumers to:

> Utilize the data encapsulated by the SWID tag to ensure that the digital signature was validated by a trusted certificate authority (CA), that the SWID tag was signed during the validity period for that signature, and that no signed data in the SWID tag has been modified. All of these validations shall be able to be accomplished without requiring access to an external network. If a SWID tag consumer needs to validate that the digital certificate has not been revoked, then it is expected that there be access to an external network or a data source that can provide [access to the necessary] revocation information.

Additional information on digital signatures, how they work, and the minimum requirements for digital signatures used for US Federal Government processing can be found in the Federal Information Processing Standards (FIPS) Publication 186-4, Digital Signature Standard (DSS) [FIPS-186-4].

2.6 A Complete Primary Tag Example

A complete tag is illustrated below, combining examples from the preceding subsections. This example illustrates a primary tag that contains all mandatory data elements as well as a number

[4] See http://www.w3.org/TR/xmldsig-core/#sec-Schema.

of optional data elements. This example does not illustrate the use of digital signatures.

```
<SoftwareIdentity
  xmlns="http://standards.iso.org/iso/19770/-2/2015/schema.xsd"
  name="ACME Roadrunner Detector 2013 Coyote Edition SP1"
  tagId="com.acme.rrd2013-ce-sp1-v4-1-5-0"
  version="4.1.5">
  <Entity
    name="The ACME Corporation"
    regid="acme.com"
    role="tagCreator softwareCreator"/>
  <Entity
    name="Coyote Services, Inc."
    regid="mycoyote.com"
    role="distributor"/>
  <Link
    rel="license"
    href=www.gnu.org/licenses/gpl.txt/>
  <Meta
    activationStatus="trial"
    product="Roadrunner Detector"
    colloquialVersion="2013"
    edition="coyote"
    revision="sp1"/>
  <Payload>
    <Directory root="%programdata%" location="rrdetector">
      <File name="rrdetector.exe" size="532712"/>
      <File name="sensors.dll" size="13295"/>
    </Directory>
  </Payload>
</SoftwareIdentity>
```

2.7 Summary

SWID tags are rich sources of information useful for identifying and describing software products installed on devices. A relatively small number of elements and attributes are required in order for a tag to be considered valid and conforming to the specification. Many other optional data elements and attributes are provided by the specification to support a wide range of usage scenarios.

A minimal valid and conforming tag uses a `<SoftwareIdentity>` element to record a product's name and the tag's globally-unique identifier, and contains an `<Entity>` element to record the name and registration identifier of the tag creator. While such a minimal tag is better than no tag at all in terms of enhancing the ability of SAM tools to discover and account for installed products, it falls short of satisfying many higher-level business and cybersecurity needs. To meet those needs, the SWID tag standard offers several additional elements, such as `<Evidence>` (for use by scanning tools to record results of the discovery process), `<Link>` (to associate tags with other items, including other tags), `<Meta>` (to record a variety of metadata values), and `<Payload>` (to enumerate files, etc., that comprise the installed product). Finally, digital signatures may optionally be used by any tag producer to ensure that the contents of a tag are not accidentally or deliberately modified after installation, and to provide authentication of the signer.

3 Implementation Guidance for All Tag Creators

The next three sections provide implementation guidance for creators of SWID tags. The primary purpose of this guidance is to help tag creators understand how to implement SWID tags in a consistent manner that will satisfy the tag-handling requirements of both public and private sector organizations. The intent of this guidance is to be broadly applicable to common IT usage scenarios that are generally relevant to IT organizations. In some limited cases, specific statements are identified as being specific to United States Government requirements. In all other cases, this guidance is directed at general usage of SWID tags.

Each guidance item in the next three sections is prefixed with a coded identifier for ease of reference from other documents. Such identifiers have the following format: *CAT-NUM*, where "CAT" is a three-letter symbol indicating the guidance category, and NUM is a number.

This section provides implementation guidance that addresses issues common to all situations in which tags are deployed and processed. Section 4 provides guidance that varies according to the type of tag being implemented (cf. Section 2.2). Section 5 provides guidance that varies according to usage scenario. Whereas Sections 3 and 4 establish minimum requirements use of SWID tags on information systems, Section 5 recognizes that SWID tags may be used for specialized business purposes, and that these specialized purposes create additional specialized tag implementation requirements.

3.1 Limits on Scope of Guidance

This document assumes that tag implementers are familiar with the SWID specification and ensure that implemented tags satisfy all requirements contained therein.

> **GEN-1.** When producing SWID tags, tag creators MUST produce SWID tags which conform to all requirements defined in the 19770-2:2015 specification.

Guidance item GEN-1 establishes a baseline of interoperability that is needed by all adopters of SWID tags.

All guidance provided in this document is intended solely to extend and not to conflict with any guidance provided by the SWID specification. Guidance in this document either:

- *Strengthens* existing guidance contained in the SWID specification by elevating "SHOULD" clauses contained in the SWID specification to "MUST" clauses, or
- *Adds* guidance where existing guidance is weak or absent by adding new "SHOULD" or "MUST" clauses to address implementation issues where the SWID specification is silent or ambiguous.

In no cases should this document's guidance be construed as either weakening or eliminating existing guidance in the SWID specification.

3.2 Authoritative and Non-Authoritative Tag Creators

SWID tags may be created by different entities (individuals, organizations, or automated tools) and under different conditions. Who creates a tag, as well as the conditions under which a tag is created, profoundly affect the quality, accuracy, completeness, and trustworthiness of the data contained in a tag.

Tags may be created by *authoritative* or *non-authoritative* entities. For the purposes of this document, an "authoritative tag creator" is defined as a 1^{st}- or 2^{nd}-party to the creation, maintenance, and distribution of the software. Essentially, any party that is involved in tag creation while releasing software is considered an authoritative tag creator. Such parties tend to possesses accurate, complete, and detailed technical knowledge of a software product at the time a tag for that product is created. Software creators are authoritative tag creators by definition.

A "non-authoritative tag creator" is defined as an entity (individual, organization, or automated tool) which is in a 3^{rd}-party relation to the creation, maintenance, and distribution of the software. Non-authoritative tag creators typically create tags using product information that is gathered indirectly, based on reverse engineering or by performing other technical analysis on the product.

Unless otherwise specified, guidance in this document is directed at both authoritative and non-authoritative tag creators. Guidance prefixed with "[Auth]" is directed specifically at authoritative tag creators, and guidance prefixed with "[Non-Auth]" is directed specifically at non-authoritative tag creators.

3.3 Implementing Required Entity Elements

Section 8.2 of the SWID specification establishes a requirement that every SWID tag contain an `<Entity>` element where the `@role` attribute has the value `"tagCreator"`, and the `@name` and `@regid` attributes are also provided.

It is important to be able to inspect a tag and rapidly determine whether the tag creator is authoritative or non-authoritative. When a tag contains only a single `<Entity>` element that describes only the tag creator role, it must be assumed that the tag creator is non-authoritative. Authoritative tag creators are required to provide one or more additional `<Entity>` elements or a single `<Entity>` element with multiple `@role` attribute values specifying organizations having any of these predefined roles: `"aggregator"`, `"distributor"`, `"licensor"`, or `"softwareCreator"`. At a minimum, authoritative tag creators must provide an `<Entity>` element identifying the `softwareCreator`.

Consumers may distinguish authoritative and non-authoritative tag creators using this rule: If the value of `<Entity> @regid` of the entity having the `@role` of `"tagCreator"` matches the value of `<Entity> @regid` of an entity having a @role value that is any of `"aggregator"`, `"distributor"`, `"licensor"`, or `"softwareCreator"`, then the tag creator is authoritative.

GEN-2. [Auth] Authoritative tag creators MUST provide an `<Entity>` element where the

`@role` attribute contains the value `softwareCreator`, and the `@name` and `@regid` attributes are also provided.

GEN-3. [Non-Auth] Non-authoritative tag creators SHOULD provide an `<Entity>` element where the `@role` attribute contains the value `softwareCreator`, and the `@name` attribute is also provided, whenever it is possible to identify the name of the entity which created the software product.

3.4 Implementing Evidence and Footprint File Data

Files are enumerated within `<Payload>` and `<Evidence>` elements using the `<File>` element. The SWID specification requires only that the `<File>` element specify the name of the file, using the `@name` attribute. Additional information is needed to enable SAM processes to check whether files have been improperly modified since they were originally deployed. By including file size information within `<Payload>` and `<Evidence>` elements using the `@size` attribute, SAM processes may rapidly and efficiently test for changes which alter a file's size. Because improper changes may occur without affecting file sizes, file hash values are also necessary.

GEN-4. Every `<File>` element provided within a `<Payload>` or `<Evidence>` element MUST include a value for the `@size` attribute that specifies the size of the file in bytes.

GEN-5. Every `<File>` element within a `<Payload>` element MUST include a hash value.

When selecting a hash function, the support lifecycle of the associated product needs to be considered. The hash value will likely be produced at the point of product release and will be used by tag consumers over the support lifecycle of the product and in some cases even longer. According to SP 800-57 Part 1 [SP800-57-part-1] when applying a hash function over a time period that extends to 2030, a minimum security strength of 112 bits is needed. A minimum security strength of 128 bits is needed if this period extends to 2031 and beyond.

Software products tend to have a support lifetime of three to five years, with use that often extends beyond this period. Stability in the hash functions used within SWID tags is also desirable to maximize the interoperability of SWID-based tools while minimizing development and maintenance costs. Taking these considerations into account, it is desirable to choose a hash function that provides a minimum security strength of 128 bits to maximize the usage period.

According to [SP800-107] the selected hash function needs to provide the following security properties:

- **Collision Resistance:** "It is computationally infeasible to find two different inputs to the hash function that have the same hash value." This provides assurance that two different files will have different computed hash values.
- **Second Preimage Resistance:** "It is computationally infeasible to find a second input that has the same hash value as any other specified input." This provides assurance that a file cannot be engineered that will have the same hash value as the original file. This makes it difficult for a malicious actor to add malware into stored executable code while maintaining

the same hash value.

Out of the FIPS 180-4 [FIPS180-4] approved hash functions, SHA-256, SHA-384, SHA-512, and SHA-512/256 meet the 128 bit strength requirements for collision resistance and second preimage resistance. This leads to the following guidance:

GEN-6. Whenever `<Payload>` or `<Evidence>` is included in a tag, every `<File>` element contained therein MUST provide a hash value based on the SHA-256 has function.

GEN-7. Whenever `<Payload>` or `<Evidence>` is included in a tag, every `<File>` element contained therein MAY additionally provide hash values based on the SHA-384, SHA-512, and/or SHA-512/256 hash functions.

Note: Use of SHA-512 may perform better on 64-bit systems.

GEN-8. Whenever `<Payload>` or `<Evidence>` is included in a tag, every `<File>` element SHOULD avoid the inclusion of hash values based on hash functions with insufficient security strength (< 128 bits).

3.5 Implementing Digital Signatures

This section contains draft guidance on the use of digital signatures within tags. Section 6.1.10 of the SWID specification discusses the use of digital signatures, and asserts no mandates for when and how signatures should be used. This section provides additional guidance to provide a reproducible, interoperable, and verifiable framework for generation and use of digital signatures.

NOTE: Guidance in this section remains to be written. NIST has found that there are interoperability concerns with the use of non-specified default values. Some canonicalization implementations do not digest these values properly.

- Question: What general requirements should be established to address this issue? Is the trust model described in NIST IR 7802 [NISTIR 7802] a suitable starting point?

- Question: How do we properly account for differences in how signing implementation handle default values when digitally signing tags? Consider requiring values for all attributes with no assumption of a default value.

3.6 Updating Tags

Section 5.2 of SWID specification requires that, once deployed, SWID tags may only be modified by the organization that initially created the tag. As the specification notes, “this is to ensure that data, especially digitally signed data, is not modified in any way that the tag producer is not directly responsible.” Nevertheless, tag creators may find it necessary from time to time to update a previously-deployed tag to correct errors or to add data elements which logically belong in the tag and not in a separate supplemental tag.

Such updating of tags can create efficiency issues if it is not easy to determine that a tag previously encountered on an endpoint has changed since it was last discovered and inspected. Tag collection and processing systems may gain significant efficiencies from analyzing tags in detail only at the time the tags are first encountered. The way this could work is that, upon encountering a tag on an endpoint, a tag processor queries a database using the tag’s `@tagId`, seeking to determine whether a tag with that tag identifier has previously been found on the endpoint. If the query result is positive (i.e., the tag was encountered previously), then no further processing is performed, otherwise, the tag is fully parsed and analyzed, and the database is updated accordingly.

To support such processing efficiencies, it is necessary to ensure that only one or two tag data elements need to be checked in order to decide whether or not the tag has been encountered previously.

> **GEN-9.** When a previously deployed tag is changed on a device, its `@tagId` attribute MUST be changed when the new tag describes a different product; e.g., the `@name` or `@version` attributes have changed.

> **GEN-10.** When a previously deployed tag is changed on a device, its `@tagVersion` attribute MUST be changed when the new tag corrects errors in the original tag.

3.7 Questions for Feedback

This section enumerates open questions related to additional implementation guidance which may be required. Feedback on these questions from reviewers is invited.

- Question: Do we need to provide guidance on tags for products which are accessible from a device (e.g., via network attached storage) rather than installed on local storage? What would such guidance look like?

3.8 Summary

These are the key points from this section:

- The primary purpose of guidance in this document is to help tag creators understand how to implement SWID tags in a manner that will satisfy the tag-handling requirements of IT organizations.

- Nevertheless, the intent of this guidance is to be broadly applicable to common IT usage scenarios that are relevant to private and commercial businesses as well.

- This section provided implementation guidance that addresses issues common to all situations in which tags are deployed and processed. The next section provides guidance that varies according to the type of tag being implemented (cf. Section 2.2).

4 Implementation Guidance Specific to Tag Type

This section provides draft implementation guidance that varies according to each of the four defined tag types (cf. Section 2.2): *primary* tags (Section 4.1), *supplemental* tags (Section 4.2), *patch* tags (Section 4.3), and *corpus* tags (Section 4.4).

4.1 Implementing Primary Tags

The primary tag for a software product contains descriptive metadata needed to support a variety of business processes. To ensure that tags contain the metadata needed to help automate IT and cybsersecurity processes on information systems, additional requirements must be satisfied. This section provides guidance addressing two topics: specification of `<Payload>` or `<Evidence>` information (Section 4.1.1), and support for mapping to Common Platform Enumeration names (Section 4.1.2).

4.1.1 Primary Tag Payload and Evidence

Detailed information about the files comprising an installed software product is a critical need. Such information enables endpoint software inventory and integrity tools to confirm that the product described by a discovered tag is, in fact, installed on a device. Thus authoritative tag creators are required to provide a `<Payload>` element, either in the primary tag or in a supplemental tag. For non-authoritative tag creators, an `<Evidence>` element needs to be provided .

> **PRI-1.** [Auth] A `<Payload>` element MUST be provided, either in a software product's primary tag, or in a supplemental tag.

> **PRI-2.** [Non-Auth] An `<Evidence>` element MUST be provided, either in a software product's primary tag, or in a supplemental tag.

Ideally, `<Payload>` and `<Evidence>` elements should list every file that is found to be part of the product described by the tag. Such information aids in the detection of malicious software attempting to hide among legitimate product files.

> **PRI-3.** `<Payload>` and `<Evidence>` elements SHOULD list every file comprising the product described by the tag.

Although a full enumeration of product files is the ideal, at a minimum, only those files subject to execution, referred to here as *machine instruction files*, need to be listed. A machine instruction file is any file that contains machine instruction code subject to runtime execution, whether in the form of machine instructions which can be directly executed by computing hardware or hardware emulators, bytecode which can be executed by a bytecode interpreter, or scripts which can be executed by scripting language interpreters. Library files that are dynamically loaded at runtime are also be considered to be machine instruction files.

> **PRI-4.** [Auth] The `<Payload>` element MUST list every machine instruction file comprising the product described by the tag.

PRI-5. [Non-Auth] The `<Evidence>` element MUST list every machine instruction file comprising the product described by the tag.

4.1.2 Mapping to Common Platform Enumeration Names

A component of NIST's Security Content Automation Protocol (SCAP), the Common Platform Enumeration (CPE) is a standardized method of naming classes of applications, operating systems, and hardware devices present among an enterprise's computing assets.[5] NIST maintains a dictionary of CPE names as part of the National Vulnerability Database (NVD).[6] Today, CPE names play an important role in the NVD, and are used to associate vulnerability reports to the affected software products. Many cyberspace defense products report discovered software using CPE names, and use those names to search the NVD for indications of vulnerability.

At some point in the future, as SWID tags become widely used and available, SWID tags will be able to supplant CPE names as the primary means of identifying software products and correlating vulnerability reports with those products. Until that occurs, SWID tags need to provide certain data values from which CPE names could be mechanically generated. These generated CPE names can be used to populate the CPE dictionary and to allow for searching repositories like the NVD. SWID tags can contain the data values in the `<Meta>` element that are needed to support CPE name generation. Four necessary `<Meta>` element attributes are:

- `@product`: This attribute provides the base name of the product (e.g., Acrobat, Creative Suite, Office, Websphere, Windows, etc.). The base name does not include substrings containing the software creator's name, or indicators of the product's version, edition, or patch/update level.

- `@colloquialVersion`: This attribute provides the informal or colloquial version of the product (e.g., 2015). Note that this version may be the same through multiple releases of a software product whereas the version specified in the `<SoftwareIdentity>` `@version` is more specific and will change for each software release.

- `@revision`: This attribute provides an informal designation for the version of the product (e.g., RC1, Beta 2, SP1).

- `@edition`: This attribute provides an informal name for a variation in a product (e.g., enterprise, personal, basic, professional).

Using these data values, a CPE name could be mechanically generated according to the following rules in Augmented BNF syntax [RFC 5234]:

---

[5] See: http://scap.nist.gov/specifications/cpe/.

[6] See: https://nvd.nist.gov/.

```
cpename = 'cpe:2.3:*:' ven ':' p ':' ver ':' u ':' e
                  ':*:*:*:*:*'
ven               = value of <Entity> @name
                    where <Entity> @role = softwareCreator
p                 = value of <Meta> @product + "_" +
                    <Meta> @colloquialVersion
ver               = value of <SoftwareIdentity> @version
u                 = value of <Meta> @revision (if not null), otherwise '*'
e                 = value of <Meta> @edition (if not null), otherwise '*'
```

For example, assume the following attribute values are provided in a tag:

- `<Entity> @name = "Fabrikam"`
- `<Meta> @product = "Office"`
- `<Meta> @colloquialVersion = "2015"`
- `<SoftwareIdentity> @version = "10.1.5"`
- `<Meta> @revision = "SP1"`
- `<Meta> @edition = "Pro"`

The following CPE name could be generated:

```
cpe:2.3:*:Fabrikam:Office_2015:10.1.5:SP1:Pro:*:*:*:*:*
```

The need for SWID tags to support such mappings to CPE names motivates the following guidance:

> **PRI-6.** A <Meta> element MUST be included in a product's primary tag. This <Meta> element MUST furnish values for the following attributes if appropriate values exist and can be determined: @product, @colloquialVersion, @revision, and @edition.

4.2 Implementing Supplemental Tags

As noted earlier (cf. Section 2.2.2), a supplemental tag is a tag where the value of the <SoftwareIdentity> @supplemental attribute is set to "true". This section provides guidance addressing two topics related to implementation of supplemental tags: the precedence of information contained in a primary tag (Section 4.2.1), and linking supplemental tags to primary tags (Section 4.2.2).

4.2.1 Precedence of Information in a Primary Tag

Supplemental tags are used to furnish data elements which complement or extend data elements furnished in a primary tag. Because all tags are required to supply a value for

`<SoftwareIdentity> @name` attribute, the possibility exists that the required value of `@name` furnished in a supplemental tag could differ from the `@name` value furnished in a primary tag. In such cases, the data value furnished by the primary tag takes precedence over the value in the supplemental tag.

SUP-1. If the `<SoftwareIdentity> @name` furnished in a supplemental tag differs from the `<SoftwareIdentity> @name` furnished in the primary tag, the value in the primary tag is considered to be the correct product name.

4.2.2 Linking a Supplemental Tag to the Primary Tag

Because the SWID specification does not clearly state how a supplemental tag should indicate its linkage to the primary tag, clarifying guidance is provided here.

SUP-2. A supplemental tag MUST contain a `<Link>` element to associate itself with the tagged product's primary tag. The `@rel` attribute of this `<Link>` element MUST be set to "`about`", and the `@href` attribute MUST be set as follows:

- **The tagId of the primary tag is known at time of supplemental tag creation:** The `@href` attribute MUST be set to a URI with "`swid:`" as its scheme, followed by the `@tagId` of the primary tag.
- **The tagId of the primary tag is not known at time of supplemental tag creation:** The `@href` attribute MUST be set to a URI reference of the primary tag, with "`swidpath:`" as its scheme, containing an XPATH query which can be resolved in the context of the system by software that can lookup other SWID tags and select the appropriate one based on an XPATH query.

4.3 Implementing Patch Tags

As noted earlier (cf. Section 2.2.2), a patch tag is a tag where the value of the `<SoftwareIdentity> @patch` attribute is set to "`true`". This section provides guidance addressing two topics related to implementation of patch tags: linking patch tags to related tags (Section 4.3.1), and specifying `<Payload>` or `<Evidence>` information (Section □).

4.3.1 Linking a Patch Tag to Related Tags

Because the SWID specification does not clearly state how a patch tag should indicate its linkage to other tags, clarifying guidance is provided here. First, a patch tag must be linked to the primary tag of each product affected by the patch. This linkage must address not only those cases where a single patch affects multiple distinct products, but also cases where a single patch affects multiple instances of the same product installed on a device.

PAT-1. [Auth] A patch tag MUST contain `<Link>` elements that associate it with the primary tag of each product instance that is affected by the patch. In such `<Link>` elements, the `<Link> @rel` attribute MUST be set to "`patches`", and the `<Link> @href` attribute MUST be set as follows:

- **The `@tagId` of the primary tag is known at time of patch tag creation:** The `@href` attribute MUST be set to a URI with "`swid:`" as its scheme, followed by the `@tagId` of the primary tag of the affected product.

- **The `@tagId` of the primary tag is not known at time of patch tag creation, or there is a need to refer to a group of tags:** The `@href` attribute MUST be set to a URI reference of the pimary tag of the affected product, with "`swidpath:`" as its scheme, containing an XPATH query which can be resolved in the context of the system by software that can lookup other SWID tags and select the appropriate one based on an XPATH query.

In some cases, a patch may *require* another patch. When a patch "B" requires another patch "A", patch A must be applied before patch B may be applied. This information must be provided to allow endpoint software inventory and integrity tools to collect a set of tags (whether primary, supplemental, or patch tags) for a given product, and then accurately determine the expected Payload on the device.

PAT-2. [Auth] A patch tag MUST contain a `<Link>` element associating it with each patch tag that describes a required predecessor patch. Each such `<Link>` element MUST have the `<Link> @rel` attribute set to "`requires`", and the `<Link> @href` attribute MUST be set as follows:

- **The `@tagId` of the required predecessor's patch tag is known at time of patch tag creation:** The `@href` attribute MUST be set to a URI with "`swid:`" as its scheme, followed by the `@tagId` of the required predecessor's patch tag.

- **The `@tagId` of the required predecessor's patch tag is not known at time of patch tag creation, or there is a need to refer to a group of tags:** The `@href` attribute MUST be set to a URI reference of the required predecessor's patch tag, with "`swidpath:`" as its scheme, containing an XPATH query which can be resolved in the context of the system by software that can lookup other SWID tags and select the appropriate one based on an XPATH query.

In other cases, a patch may *supersede* another patch. When a patch "B" supersedes patch "A", it effectively implements all the changes implemented by patch A. This information must be provided to allow scanning tools to accurately determine an expected Payload.

PAT-3. [Auth] A patch tag MUST contain a `<Link>` element associating it with each patch tag that describes a superseded patch. Each such `<Link>` element MUST have the `<Link> @rel` attribute set to "`supersedes`", and the `<Link> @href` attribute MUST be set as follows:

- **The `@tagId` of the superseded patch tag is known at time of patch tag creation:** The `@href` attribute MUST be set to a URI with "`swid:`" as its scheme, followed by the tagId of the superseded patch tag.

- **The `@tagId` of the superseded patch tag is not known at time of patch tag creation, or there is a need to refer to a group of tags:** The `@href` attribute MUST be set to a URI reference of the required predecessor's patch tag, with "`swidpath:`" as its scheme, containing an XPATH query which can be resolved in the context of the system by software that can lookup other swidtags and select the appropriate one based on an XPATH query.

4.3.2 Patch Tag Payload and Evidence

Patches change files that comprise a software product, and may thereby eliminate known vulnerabilities. If patch tags clearly specify the files that are changed as a result of applying the patch, software inventory and integrity tools become able to confirm that the patch has actually been applied, and that the individual files discovered on the endpoint are the ones that should be there.

This guidance proposes that patch tags document three distinct types of change:

1. **Change:** A file previously installed as part of the product has been modified on the device.
2. **Remove:** A file previously installed as part of the product has been removed from the device.
3. **Add:** An entirely new file has been added to the device.

For files that are changed or added, patch tags must include file size and hash values. Authoritative tag creators are required to provide this information in the `<Payload>` element of the patch tag. Non-authoritative tag creators are encouraged to provide this information whenever possible in the `<Evidence>` element of the patch tag.

PAT-4. [Auth] A patch tag MUST contain a `<Payload>` element which MUST enumerate every file that is changed, removed, or added by the patch.

PAT-5. [Auth] Each `<File>` element contained within the `<Payload>` element of a patch tag MUST include an extension attribute named `@patchEvent`, which MUST be one of the following values:

- The string value "`change`" to indicate a preexisting file has been modified on the device
- The string value "`remove`" to indicate a preexisting file has been removed from the device
- The string value "`add`" to indicate a new file has been added to the device

PAT-6. [Non-Auth] A patch tag MUST contain an `<Evidence>` element which enumerates every file that was used as part of the detection process.

4.4 Implementing Corpus Tags

As noted earlier (cf. Section 2.2.2), a corpus tag is a tag where the value of the `<SoftwareIdentity> @corpus` attribute is set to "`true`". This section provides guidance addressing two topics related to implementation of corpus tags: specification of Payload information (Section 4.4.1), and signing of corpus tags (Section 4.4.2).

4.4.1 Corpus Tag Payload

Corpus tags are used to document the installation media associated with a software product. This documentation enables the media to be checked for authenticity and integrity. The usual distinction between authoritative and non-authoritative tag creators does not apply to creators of corpus tags. The creator of installation media for a given software product may, but need not be, the same entity that created the product itself. Any creator of installation media is considered to be an authoritative tag creator of any associated corpus tag. Furthermore, it is expected that any creator of a corpus tag must necessarily have sufficient access to the installation media being tagged to be able to satisfy the guidance below.

At a minimum, corpus tags are required to provide Payload details that enumerate all the files on the installation media, including file size and hash values.

> **COR-1.** A corpus tag MUST contain a `<Payload>` element which MUST enumerate every file that is included in the tagged installation media.

4.4.2 Corpus Tag Signing

As noted above, corpus tags are needed to support authenticity and integrity checks. For this to work, the tags themselves must be digitally signed to ensure that the data values contained within the tag, including the `<Payload>` details, have not been modified, and a separate signature is required to support authentication of the provider of the tag.

- <u>Question</u>: What is the appropriate guidance to provide w/r/t signing of corpus tags?

4.5 Summary

This section provided draft implementation guidance related to all four SWID tag types: primary, supplemental, patch, and corpus. Key points:

- Authoritative creators of primary tags are required to provide `<Payload>` information, and to include `<Meta>` attribute values needed to support automated generation of Common Platform Enumeration names. Non-authoritative creators of primary tags are required to provide `<Evidence>` information for any data used to detect the presence of the product.

- Any value supplied for `<SoftwareIdentity> @name` in a supplemental tag is overridden by the value supplied for `<SoftwareIdentity> @name` in the primary tag. Supplemental tags must provide `<Link>` information associating them with the primary tag.

- Patch tags must be explicitly linked to the primary tag of the patched product, as well as to any tags of required predecessor patches or superseded patches. Patch tags must document all files changed, removed, or added by the patch.
- Corpus tags must include `<Payload>` details, and must be digitally signed to facilitate authentication and integrity checks.

5 SWID Tag Usage Scenarios

This section describes a number of usage scenarios for software asset management and for software integrity management. These are not intended to represent an exhaustive or conclusive list of possible SWID applications, but provide informative examples regarding the use of the SWID specification to accomplish various organizational needs.

5.1 Software Inventory Management

Proper understanding and control of the software deployed on devices within the organization enables network security professionals to achieve security requirements. Software Asset Management (SAM) helps to ensure effective management of software assets, including the identification of potential software weaknesses that may be exploited. SAM is an important component of planning and execution for system backup and recovery processes. The use of SWID tags as described in the previous sections provides for interoperability and automation supported by a variety of situational awareness and configuration management products. These products, for example, evaluate the difference between the observed software inventory (from SWID tags) and a desired state specification. Continuous monitoring processes can use the SWID tag data to identify and report any variance, such as in the examples below. The use of SWID tags also reduces reliance on proprietary algorithms used by commercial-off-the-shelf (COTS) products for identifying installed applications, software components, and patches within an IT environment.

5.1.1 Usage Scenario 1 – Collecting Software Inventory Information from an Endpoint

A primary usage of SWID tools is to enable automated tools to collect information from an organization’s endpoints, building a comprehensive inventory of the installed software products on each endpoint and supporting effective search and analysis. This type of usage can support operational decisions by indicating if a software product is authorized for use, meets licensing requirements, and has been properly patched against vulnerabilities.

SWID tags are portable across different device types and platforms. Both SWID tags and the data they represent may be stored in a local repository on the endpoint, or may be recorded centrally by an enterprise system. This data may be updated periodically (e.g. every 72 hours), or as needed to support an event-based requirement.

The use of standardized data and tagging implementation models provided by SWID tags for deployed software enables tools to easily share software inventory information and “roll-up” software inventory reports.

5.1.1.1 Assumptions

This usage scenario assumes that the following conditions exist:

- The discovery tool has sufficient access rights to the endpoint to discover each software instance and the metadata about it
- The discovery tool has network connectivity to the endpoint

5.1.1.2 Process

The SAM tool acquires the complete set of tags from each endpoint via its own agent installed locally on the managed system, or via a remote management interface that can collect the SWID tags. Additionally, the SAM tool gathers endpoint identification information (host name, IP addresses, etc.), the date/time of the data collection, and data about the discovery tool agent or remote management interface used.

For each managed system in the local and/or central repository:

1. Update the inventory database with the data from the existing SWID tags creating entries for software products and their components. At a minimum, those tags SHOULD include the software `@name` and `@version` attribute values of the `<SoftwareIdentity>` element. If the version scheme is not the commonly-used multipart-numeric scheme (e.g., has a suffix such as 1.2.3a), the tag SHOULD use the `@VersionScheme` attribute to indicate the encoding method used.

2. Record additional information contained within the SWID tags. The information below SHOULD be collected, if available:
   - Values from `<Payload>` element, `@File` attributes such as name, size, location, and cryptographic algorithm/hash.
   - Information from `<Link>` elements that describe a relationships to another software item or additional product data (e.g. licensing information) through the `<Meta>` elements.

3. If a tag has not been installed with the software, the SAM tool will create a 3rd -party tag on the endpoint for each instance of an application discovered. That 3rd -party tag will include relevant data using the `<Evidence>` element about the software products installed. This information SHOULD include data from the `<Evidence>` element, `@File` attributes such as file name, size, location, and cryptographic algorithm/hash.

5.1.1.3 Outcomes

Through the use of SWID tags for software inventory collection, organizations are able to improve situational awareness through more accurate and timely discovery of software data. This supports *software inventory management* as described above —the process of building and maintaining an accurate and complete inventory of all software products deployed on all of the devices under an organization's operational control.

5.1.2 Usage Scenario 2 –Software Inventory Reporting

Based on data previously collected, as described in Section 5.1.1, SWID tags enable many software reporting capabilities regarding the software inventory of enterprise systems. SWID tags enable accurate and reliable reporting of the software products installed on endpoints within the infrastructure, and exchange of relevant data about those products. Together, this information is critical in effectively managing information technology across an enterprise. SWID tags provide a vendor-neutral and platform-independent way to report software installation state (e.g. software installed, products missing, or applications in need of patching.) Several example processes are described below, representing a subset of the potential reporting capabilities enabled by the use of SWID tags.

5.1.2.1 Assumptions

This usage scenario assumes that the following conditions exist:

- A software asset management repository is populated with SWID tags from a given endpoint and will be updated on a timely or event-driven basis as the endpoint software inventory changes.
- At a minimum, those tags SHALL include the software `@name` and `@version` attribute values of the `<SoftwareIdentity>` element.

5.1.2.2 Process 1: Reporting the Software Installed on an Endpoint

1. For a given endpoint, the SAM Tool iterates through each tag in the repository including 3rd-party SWID tags.
2. The SAM Tool parses the values contained in `@name` and `@version` attribute of the `<SoftwareIdentity>` element and other relevant software identification information (e.g. revisions, colloquial names) to create an accurate and comprehensive report of the software discovered.
3. The software inventory report is provided through the SAM Tool's dashboard and/or reporting process. As appropriate, the SAM Tool may trigger alerts based on pre-determined conditions (e.g. prohibited software detected.)

5.1.2.3 Process 2: Identifying Instances of a Given Product

One common enterprise need is to determine which endpoints have a specific product installed, such as to confirm that a mandatory software item and version are installed. Consider a scenario where we want to report the endpoints that contain the product, Acme Roadrunner, and the versions installed on those endpoints.

1. For a given endpoint (or set of endpoints), the SAM Tool iterates through each tag in the repository including 3rd-party SWID tags.
2. The SAM Tool parses the values contained in `@name` and `@version` attributes of the `<SoftwareIdentity>` element, searching specifically for values for `@name` = "Acme Roadrunner". Where a match is located, the SAM Tool records the endpoint identifier for the device on which the tag was found and notes relevant version information from the values for the `@version` attribute.

3. The software inventory report is provided through the SAM Tool's dashboard and/or reporting process. As appropriate, the SAM Tool may trigger alerts based on pre-determined conditions (e.g. prohibited software detected.)

5.1.2.4 Process 3: Identifying Endpoints That Are Missing a Product

Another common need is to determine which endpoints are missing a required software product. Consider a scenario where the implementation baseline requires each endpoint to contain the product, Acme Roadrunner, version 12.2.

1. Through a dashboard or other internal process, the SAM Tool is informed about the endpoint (or set of endpoints) that are required to contain the referenced software product and version. The SAM Tool iterates through the recorded tags in the repository, including 3rd-party SWID tags, associated with that set of one or more endpoints.

2. The SAM Tool parses the values contained in `@name` and `@version` attributes of the `<SoftwareIdentity>` element, searching specifically for values for `@name` = "Acme Roadrunner" and the value "12.2" from the `@version` attribute.

3. Where a match is not located, the SAM Tool records the endpoint identifier for each device that does not comply with the requirement from Step 1. Optionally, where a match is located, the SAM Tool records the endpoint's compliant state.

4. The software inventory report is provided through the SAM Tool's dashboard and/or reporting process. As appropriate, the SAM Tool may trigger alerts based on pre-determined conditions (e.g. required software determined to be absent.)

5.1.2.5 Process 4: Identifying Endpoints That Contain or are Missing a Patch

Product providers often create software patches, such as to improve performance, introduce a new feature, or mitigate a vulnerability. Consumers often need reports about endpoints that are missing a known patch for security awareness or to help prepare installation plans.

1. For a given endpoint (or set of endpoints), the SAM Tool iterates through each tag in the repository including 3rd-party SWID tags.

2. As described in Section 5.1.2.3, the SAM Tool parses the values contained in `@name` attribute of the `<SoftwareIdentity>` element, searching specifically for values where the `@name` matches the patch tag name.

3. Where a match is not located, the SAM Tool records the endpoint identifier for the unpatched device. Optionally, where a match is located, the SAM Tool records that fact.

4. The software inventory report is provided through the SAM Tool's dashboard and/or reporting process. As appropriate, the SAM Tool may trigger alerts based on pre-determined conditions (e.g. endpoints that are missing a given security patch.)

5.1.2.6 Process 5: Identifying Orphaned Software Components/Patches on Endpoints

Components of previously installed software products, including patches that were applied but left behind when that product was uninstalled, might use valuable resources on an endpoint. These orphaned components may also represent a software vulnerability if they contain an exploitable flaw. SWID tag reporting can identify endpoints that contain items such as binaries and runtime libraries that belong to no installed package.

1. For a given endpoint (or set of endpoints), the SAM Tool iterates through each tag in the repository including 3rd-party SWID tags. The Tool specifically inspects tags indicating relationships to other products as indicated by the `<Link>` element, `@rel` attribute. (e.g., a SWID tag for the French Language Pack for RoadRunner Word Processor identified by `tagId="{GUID}RoadRunnerWP-2013-French"` with a value of `"parent"` in the `@rel` attribute of the `<Link>` element and pointing to `@href` value `"swid:{GUID}RoadRunnerWP-2013"`)

2. For each such tag located, the SAM Tool verifies the installation of the parent software by checking for the referenced installation SWID tag (in this example, `"swid:{GUID}RoadRunnerWP-2013"`)

3. Where a match is not located, the SAM Tool records that an orphaned software component may exist on that endpoint.

4. The software inventory report is provided through the SAM Tool's dashboard and/or reporting process.

5.1.2.7 Process 6 – Reporting Installation of Authorized or Prohibited Software

Many organizations strictly control what software may or may not be installed on information systems. SAM tools, supported by collected SWID tag information, can provide specific reports that confirm that all installed software on a given endpoint matches the specification of an "approved software baseline", or whitelist. Often, this comparison will be based upon evaluation of the name and version information from the <SoftwareIdentity> element, @name and @version attributes.

1. Through a dashboard or other internal process, the SAM Tool is provided with a set of SWID tags that represent (a) a list of approved software items (i.e., a "whitelist"), or (b) a list of prohibited software items (i.e., a "blacklist".)

2. The SAM Tool iterates through the recorded tags in the repository, including 3rd-party SWID tags, associated with one or more endpoints on which to report.

3. The SAM Tool parses the values contained in `@name` and `@version` attributes of the `<SoftwareIdentity>` element, searching specifically for values in the `@name` attribute and optionally from the `@version` attribute. The tool compares each value to the list provided in step 1.

4. If additional confirmation is required, such as to help prevent against an unauthorized product masquerading as approved software, the SAM tool can compare the observed cryptographic hash of each software product (from the `<Payload>` element, `@File` attribute, cryptographic algorithm/hash, stored in the SWID tag) with hash values stored in the listing from step 1 (the “whitelist” or “blacklist”).

5. Where a match to an authorized software product is <u>not</u> located, the SAM Tool reports that condition. This information may support a security policy decision such as whether to only permit a network connection from a device with a required anti-virus product.

6. Where a match to a blacklisted software product <u>is</u> located, the SAM Tool reports that condition. This information may support another type of security policy decision, such as quarantining a device that is found to contain a software product that is specifically prohibited.

7. The SAM Tool’s may also perform other reporting such as sending logs or alerts to a Security Information Event Management (SIEM) system.

5.1.2.8 Outcomes

For each of the processes described above, the application of SWID tags enables the organization to use automation for the accurate and timely reporting of software inventory information. While many of these processes are achievable without SWID tags, the consistent and precise information these tags provide is beneficial.

5.2 Usage Scenario 3 – Determining Vulnerable Software on an Endpoint

SWID tags provide valuable information to relate software installation information with vulnerability findings from one or more sources (described below). Vulnerability assessment is performed to identify flaws in an endpoint’s software. If an endpoint’s software is updated in a timely fashion and has no unmitigated known vulnerabilities, no action is needed; unfortunately, usually that’s not the case. SWID tags provide comprehensive, compact description of software installed which may then be compared with a source of vulnerability information to automatically find vulnerabilities. Without SWID tags, it is necessary to examine all the endpoints to determine potentially vulnerable software. Through the use of a consistent and standardized structure, SWID enables effective operations between the vulnerability information sources (e.g. National Vulnerability Database, vendor alerts, US CERT alerts) and the SAM tools that collect inventory information.

5.2.1.1 Assumptions

This usage scenario assumes that the following conditions exist:

- A software asset management repository will be populated with SWID tags from a given endpoint and will be updated on a timely or event-driven basis as the endpoint software inventory changes.
- At a minimum, those tags SHALL include the software `@name` and `@version` attribute values of the `<SoftwareIdentity>` element.
- If a tag has not been installed with the software, a SAM tool will have created a 3rd party tag for each instance of an application discovered on the endpoint. That 3rd party tag will include relevant data (using the `<Evidence>` element) about the software products installed. It should be noted that the accuracy and completeness of such inventory tags will be limited if the discovery tool does not have sufficient access rights to the endpoint.

5.2.1.2 Process 1 – Including SWID Tag Information in a Vulnerability Bulletin

Many software providers create occasional bulletins that describe vulnerabilities that have been discovered within software products. These bulletins SHOULD include SWID tag information to uniquely describe vulnerable software as follows:

1. The vulnerability bulletin SHOULD provide name and version information which can be used by SAM tools to compare with endpoint tag data. At a minimum, that data SHOULD include information that will match the software `@name` and `@version` attribute values of the `<SoftwareIdentity>` element. If the version scheme is not the commonly-used multipart-numeric scheme (e.g., has a suffix such as 1.2.3a), the bulletin SHOULD use the `@versionScheme` attribute to indicate the encoding method used.
2. If a *software provider* uses additional information to identify the software product (e.g. Professional Edition), this additional data MUST be included in the bulletin to match SWID tag data, using the `<Meta>` element providing at least the `@product`, `@productFamily`, and `@revision` attributes.

5.2.1.3 Process 2 – Use of SWID Tag Data for Determining Vulnerable Software

1. Using the information about reported software vulnerability from one or more software vulnerability bulletins, the SAM tool reviews each SWID tag record.
2. Where a record exists that matches the `<SoftwareIdentity>` element, `@name`, `@version`, and `@versionScheme` attributes, the associated endpoint is flagged as containing vulnerable software.

3. Where patch SWID tag information is provided in the bulletin, the SAM tool queries the database to determine whether the appropriate patch tag has been installed.

4. If the endpoint is found to contain vulnerable software but not the associated patch, the system may be flagged to support other potential mitigation activities.

> Consider the case of the vulnerability described by a fictional CVE, CVE-1990-0301. It describes a known buffer overflow in the product named Acme Roadrunner, versions between 11.1 and 12.1. The issue was remediated in version 12.2 and later. There is also a patch KB123 that mitigates the vulnerability. The SAM tool can use matching logic to review the collected SWID tags for the endpoint, searching for installed software instances that match:
>
> SoftwareIdentity> @name=”Acme Roadrunner” and either:
> whose major version is 11 and minor version is greater than or equal to 1; or
> whose major version is 12 and minor version is less than 2.
>
> And also the presence of the following in the software inventory:
> <SoftwareIdentity> @name=”Acme_Roadrunner_KB123”.
>
> Upon discovering a SWID tag that indicates the installation of a vulnerable version of the Acme Roadrunner product (e.g. Acme Roadrunner version 11.5), the SAM tool searches through the repository and discovers a Patch Tag named “Acme_Roadrunner_KB123” associated with that endpoint.
>
> Given the above scenario, the SAM tool reports that the endpoint contains software with a known vulnerability, but appears to have been patched. This information can be reported for security situational awareness and supports security analysis.

5.2.1.4 Outcomes

Through the use of SWID tags for the description and discovery of vulnerable software, organizations are able to achieve accurate and timely security situational awareness.

5.3 Software Integrity Management

SWID tags support an organizations ability to identify signs that a software product may have been tampered with, such as through comparison of the current cryptographic hash with that recorded previously. This information may be used to help prevent execution of an application where tampering is suspected, or to alert a security reporting process.

5.3.1 Usage Scenario 4 - Detection of software tampering

An important element of software asset management is the discovery of any files on endpoints that have been tampered with since the software was installed. This condition may be part of a SAM report, or may be used by a security product to quarantine or prevent execution of an application that shows signs of tampering.

Organizations are encouraged to take advantage of this capability using SWID tags to convey important information about the characteristics of installed software. Specifically, the ability to store and compare cryptographic hashes of installed executable software is a useful method to identify potential tampering or unauthorized changes.

5.3.1.1 Assumptions

This usage scenario assumes that the following conditions exist:

- A software asset management repository will be populated with SWID tags from a given endpoint and will be updated on a regular basis as the endpoint software inventory changes. If a tag has not been installed with the software, a SAM tool has created a 3rd party tag for each instance of an application discovered on the endpoint
- An organization has chosen to use the SWID tag cryptographic hash capabilities to detect tampering or other unauthorized changes.
- The SAM tool records a cryptographic hash for each executable file on each endpoint by recording each hash in `<Payload>` element, `@File` attribute, cryptographic algorithm/hash value.

5.3.1.2 Process

1. For each endpoint, the SAM tool reads the stored cryptographic hashes for each file listed in `<Payload>` element, `@File` attribute, cryptographic algorithm/hash.
2. The SAM tool calculates the current cryptographic hash of the actual files on those endpoints, using the same algorithm as originally used in the SWID tags.
3. If any file hash does not match the manifest provided, the reporting tool will set an error condition that will report the variance and/or help prevent that application from being used. Note: this operation is likely to result in high utilization of the resources on those endpoints and should be performed with caution.

5.3.1.3 Outcomes

Identifying tampered executable files in an automated, accurate and timely manner supports an organization's ability to prevent execution of files that have been infected by malware or by other types of malicious activities.

5.4 Usage Scenario 5 - Mapping SWID Tag to Other SWID Schemes

Many software identification schemes exist today, some standardized and others proprietary. The data provided within SWID tags can support automatic translations to other schemes (e.g., CPE). SWID can also provide stable identifiers and categorization data that can be used to creating mappings.

The primary use cases for this category include:

- Legacy systems and tools that rely upon the use of CPE and are not planning to change to SWID in the near future
- Systems and tools that are in the process of migrating from CPE to SWID and must support both during some transition timeframe.

5.5 Usage Scenario 6 - Network-Based Policy Enforcement based on SWID Information

Controlling access to network resources enables organizations to ensure that the state of an endpoint is acceptable at the time of connection and on an ongoing basis. Detecting and evaluating the software inventory of a device, based on SWID tags, is an important dimension of network access control decisions.

Appendix A—Acronyms

Selected acronyms and abbreviations used in this paper are defined below.

CPE	Common Platform Enumeration
ISCM	Information Security Continuous Monitoring
NVD	National Vulnerability Database
SCAP	Security Content Automation Protocol
USG	United States Government

Appendix B—References

[ISO/IEC 19770-2:2009]	International Organization for Standardization/International Electrotechnical Commission, *Information technology -- Software asset management -- Part 2: Software identification tag*, ISO/IEC 19770-2:2009, 2009. http://www.iso.org/iso/catalogue_detail?csnumber=53670 [accessed 5/26/15].
[RFC 3986]	Berners-Lee, T., Fielding, R., and Masinter, L. (2005). *Uniform Resource Identifier (URI): Generic Syntax*. Internet Engineering Task Force (IETF) Network Working Group Request for Comments (RFC) 3986, January 2005. https://www.ietf.org/rfc/rfc3986.txt [accessed 5/26/15].
[XAdES]	Cruellas, J., Karlinger, G., Pinkas, D., and Ross, J. (2003), *XML Advanced Electronic Signatures (XAdES)*. World Wide Web Consortium (W3C) Note, February 2003. http://www.w3.org/TR/XAdES/ [accessed 5/26/15].
[xmldisg-core]	Bartel, M., Boyer, J., Fox, B., LaMacchia, B, and Simon, E. (2008), *XML Signature Syntax and Processing (Second Edition)*. World Wide Web Consortium (W3C) Recommendation, June 2008. http://www.w3.org/TR/xmldsig-core/ [accessed 5/26/15].
[FIPS186-4]	U.S. Department of Commerce. *Digital Signature Standard (DSS)*, Federal Information Processing Standards (FIPS) Publication 186-4, July 2013, 130pp. http://dx.doi.org/10.6028/NIST.FIPS.186-4 [accessed 5/26/15].
[SP800-57-part-1]	NIST Special Publication (SP) 800-57 Part 1 Revision 3, *Recommendation for Key Management – Part 1: General*, National Institute of Standards and Technology, Gaithersburg, Maryland, July 2012, 147pp. http://csrc.nist.gov/publications/nistpubs/800-57/sp800-57_part1_rev3_general.pdf [accessed 5/26/15].
[SP800-107]	NIST Special Publication (SP) 800-107 Revision 1, *Recommendation for Applications Using Approved Hash Algorithms*, National Institute of Standards and Technology, Gaithersburg, Maryland, August 2012, 25pp. http://csrc.nist.gov/publications/nistpubs/800-107-rev1/sp800-107-rev1.pdf [accessed 5/26/15].
[FIPS180-4]	U.S. Department of Commerce. *Secure Hash Standard (SHS)*, Federal Information Processing Standards (FIPS) Publication 180-4, March 2012, 37pp. http://csrc.nist.gov/publications/fips/fips180-4/fips-180-4.pdf [accessed 5/26/15].
[NISTIR 7802]	Booth, H., and Halbardier, A. (2011). *Trust Model for Security Automation Data 1.0*.National Institute of Standards and Technology Interagency Report 7802. Available at: http://csrc.nist.gov/publications/nistir/ir7802/NISTIR-

	7802.pdf. [accessed 5/26/15].
[RFC 5234]	Crocker, D., and P. Overell. (2008). *Augmented BNF for Syntax Specifications: ABNF*. Internet Engineering Task Force (IETF) Request for Comments (RFC) 5234, January 2008. https://tools.ietf.org/html/rfc5234 [accessed 5/26/15].