NISTIR 8060 (Second DRAFT)

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

NISTIR 8060 (Second DRAFT)

Guidelines for the Creation of Interoperable Software Identification Tags

David Waltermire
Computer Security Division
Information Technology Laboratory

Brant A. Cheikes
Cyber Security Technical Center
The MITRE Corporation
Bedford, Massachusetts

July 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
72 pages (July 2015)

Public comment period: *July 22, 2015* through *August 7, 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 report provides an overview of the capabilities and usage of software identification (SWID) tags as part of a comprehensive software lifecycle. As instantiated in the International Organization for Standardization (ISO)/International Electrotechnical Commission (ISO/IEC) 19770-2 standard, SWID tags support numerous applications for software asset management and information security management. This report introduces SWID tags in an operational context, provides guidelines 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 (SWID); software identification tag

Acknowledgments

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

Note to Reviewers

This document represents a second discussion draft of this report. The authors are conducting a number of iterations of this document to further develop the concepts and guidelines 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 a total of three to six iterations of this cycle before finalizing this document. While this is a slight departure from the normal development cycle for a NISTIR, the authors believe that this collaborative approach will result in a better set of usable guidance for SWID tag creators.

For this draft iteration, review should be focused on the overall document, especially the requirements defined in sections 3 and 4 of this report. Specific attention should be given to any inline questions in the report. These questions represent areas where feedback is needed to complete this 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

List of Figures

List of Tables

1 Introduction

International Organization for Standardization (ISO)/International Electrotechnical Commission (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 that 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 ISO/IEC 19770-2:2015 version of SWID tags, focusing on the use of SWID tags as part of comprehensive software asset management lifecycles and cybersecurity procedures.

Section 1.1 discusses the software asset management and cybersecurity problems that motivated the development of SWID tags. Section 1.2 highlights the significant benefits that 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 report. Section 1.4 summarizes this section's key points, and Section 1.5 describes how the rest of this report 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 19770-5 [ISO/IEC 19770-5:2013], a companion standard to the SWID specification, defines *software asset management* (SAM) as "control and protection of software and related assets within an organization, and control and protection of information about related assets which are needed in order to control and protect software assets." 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 lifecycle 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 that 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 creators of software products as well as 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 that accompany their products. This is 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, 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 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 are enabled to consume the associated tags. This gives rise to 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 inventory, licensing, cybersecurity, or 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 ongoing basis, including software creators, software licensors, packagers, and 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 report has three purposes. First, it provides a high-level description of SWID tags, in order to increase familiarity with the standard. Second, it provides guidelines that supplement the SWID tag specification pertaining to the creation of specific types of SWID tags. 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 helps 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 that 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 that 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 report helps software consumers understand the benefits of software products that 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 report 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

The following 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 that create tags as well as to those that consume tags.
- Three audiences have interrelated goals related to SWID tags and tagging practices:
    - *Software providers* may want to increase the manageability of their products for their customers. To justify investing 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* may want 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.
    - *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 supply tags along with their products as a common practice.
- This document seeks to raise awareness of the SWID tag standard, promote understanding of the business and cybersecurity benefits that 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 Report Structure

The remainder of this report 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 guidelines that address issues common to all situations in which tags are deployed and processed on information systems. The intent of these guidelines is to be broadly applicable to common IT usage scenarios that are relevant to both public and private sector organizations.
- Section 4 provides implementation guidelines that vary according to the type of tag being implemented.
- Section 5 describes several usage scenarios for software asset management and software integrity management. These are not intended to represent an exhaustive or conclusive list of possible SWID applications; they provide informative examples regarding the use of the SWID specification to accomplish various organizational needs.
- Appendix A describes a mechanical procedure for forming Common Platform Enumeration (CPE) names using SWID tag data elements.
- Appendix B presents a list of selected acronyms used in this report.

- Appendix C provides the references for the report.
- Appendix D details the change log for the report.

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. Software products and their tags are logically separate entities. When a software product is installed on a computing device, one or more SWID tags associated with that product can be installed or otherwise become discoverable on that device. When a product is uninstalled from a device, all associated tags are expected to be removed.[1] When software is upgraded, any SWID tags representing the old software version are expected to be replaced with one or more SWID tags for the newer version. In this way, the presence of a tag on a device serves as evidence of the presence on that device of the related software product and product version described by the tag. The SWID specification defines these behaviors, as well as related behaviors associated with software licensing, patching, and upgrading. For cases where a software product is installed on a device, and one or more tags describing that product are discoverable on the device, this document uses the term *tagged software product* (or, simply, *tagged product*) to refer to the product.

Section 5.2 of the SWID specification states that once a SWID tag has been installed on a device, the contents of that tag may be modified only by "the organization that initially created the tag," i.e., the tag creator. Furthermore, the specification requires that every SWID tag identify the tag creator in the tag's `<Entity>` element (see Section 2.4.2 of this document). This restriction is necessary to ensure that any supplied digital signatures and thumbprints used to authenticate SWID tags remain valid and usable (see Section 2.5). Nevertheless, because there is a recognized need for additional identifying and/or descriptive data to be furnished at different times by different parties, the SWID specification defines a special mechanism for that purpose—the supplemental tag (see Section 2.2.4).

This section presents a high-level description of SWID tag data elements as specified in the SWID specification. The material presented here is intended to provide a general understanding of 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 SWID tags and the distinct roles they play at key points in the SAM lifecycle. Section 2.3 discusses three main ways in which SWID tags are deployed to devices. Section 2.4 presents an overview of the basic data elements that comprise a

[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.

SWID tag. Section 2.5 discusses how SWID tags may be authenticated. Section 2.6 presents an example of the primary tag type, and Section 2.7 concludes with a summary of key points from this section.

2.1 SWID Tag Placement

This section discusses where SWID tags are placed relative to the products that they identify and describe. The SWID specification makes the following statements about SWID tag placement:

> On devices with a file system, but no API defined to retrieve SWID tags, the SWID tag data shall be stored in an XML file and shall be located on a device's file system in a sub-directory named "swidtag" (all lower case) that is located in the same file directory or sub-directory of the install location of the software component with which they are installed. It is recommended, but not required, that the swidtag directory is located at the top of the application installation directory tree. Any payload information provided must reference files using a relative path of the location where the SWID tag is stored. On devices that do not have a file system, the SWID tag data shall be stored in a data storage location defined and managed by the platform provider for that device. […] On devices that utilize both a file system for software installation as well as API access to the SWID tag files, it is recommended that the SWID tag data be stored in the API managed repository as well as stored as a file on the system. […] Finally, the SWID tag data may also be accessible via a URI, or other means […] [ISO/IEC 19770-2:2015, pp. 7-8].

These statements suggest that the SWID tag for a product is placed on the same device where the product is installed. While this is correct as a general rule, as the IT market has evolved, the concept of an "installed software product" has become increasingly nuanced, and this has complicated the issue of where SWID tags may be placed.

The simplest concept of an "installed software product" is software that can 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 not always true. 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 co-reside on the NAS device, and inventory tools will likely consider the products to be part of the inventory of the NAS device. In other words, storage location matters more than the location where a product can be executed

[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.

when determining tag placement. The locations where a product can be executed may need to be considered, however, when determining the effective software inventory of an endpoint.

As another example, consider removable media devices such as USB thumb drives and SD memory cards. Once a software product is installed on such removable media, it can become executable on an endpoint immediately upon insertion of the media. In this scenario, the product tag resides with the product on the removable media. The product is considered part of the inventory of the removable media, but may also be considered part of the effective software inventory of the endpoint during the time the removable device is attached.

The rise of virtualization technology further clouds the issue, as it changes the definition of what it means to be a computing device, and introduces the prospect of virtual devices that are created, inventoried, and destroyed all in the space of mere moments. In general, SWID tags for software products that are installed on virtual machines reside within the virtual machine images, and are accountable to the virtual machines rather than to the physical host machines. When software products are installed on a virtual machine that is powered down, inactive, and stored somewhere as a machine image, those products are considered to exist in the inventory of the virtual machine, not the inventory of the device that stores the machine image. In this sense, a powered-down virtual machine is treated no differently than a powered-down physical machine. Similarly, destroying a virtual machine is treated no differently than decommissioning a physical machine. Software products and their associated tags would be removed from inventory in both cases.

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 rely on short-lived software, often executed in a browser, which breaks the linkage between where products are installed and where they are executed. When a software application is operated remotely as a service, it is 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 reside on the same physical or virtual storage device as where the tagged product resides. Although tag consumers may infer that a product is executable on the same device where it is stored, they will benefit from distinguishing cases where products may be executable on devices elsewhere within the enterprise.

2.2 SWID Tag Types and the Software Lifecycle

The SWID specification defines four types of SWID tags: *corpus*, *primary*, *patch*, and *supplemental*. Corpus, primary, and patch tags have similar functions in that they describe the existence and/or presence of different types of software, and potentially also different states of software products. These three tag types come into play at different points in the software lifecycle, and they support software management processes that depend on the ability to accurately determine where each software product is in its lifecycle. These are the key points in the software lifecycle that are supported by these three types of SWID tags:

- **Receipt of a software installation package.** Before software is installed, it is typically delivered or otherwise made available to an endpoint in the form of an installation package. The installation package contains the software in a pre-installation condition, often compressed in some manner. Common formats for installation packages include TAR and ZIP files, and “self-unpacking” executable files. In all cases, an installation procedure must be run to cause the software contained in an installation package to be unpacked and deployed on a target endpoint. Corpus tags are used to identify and describe the software contained in an installation package in its pre-installation state. Corpus tags are discussed in detail in Section 2.2.1.

- **Installation of software on an endpoint.** Upon completion of the pertinent software installation procedure on a given endpoint, a primary tag is deployed (or otherwise made available through a platform-specific interface) to identify and describe the software in its post-installation state on the endpoint. Primary tags are discussed in detail in Section 2.2.2.
- **Application of a software patch.** 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 repair defects in software products having large and complex codebases, such as operating systems and major applications. When a tagged product is patched, a patch tag is installed as part of the patch procedure. Usually, if a patch is uninstalled, the associated patch tag is also removed. Patch tags are discussed in detail in Section 2.2.3.
- **Application of a software upgrade.** When a software product is upgraded, major changes are made to the product’s codebase, often necessitating a change in the product’s version number. Software upgrades are reflected by removing (or archiving) all tags associated with the pre-upgrade version, and deploying one or more new tags to identify and describe the post-upgrade version.
- **Removal of software from an endpoint.** When a software product is no longer needed or wanted, it is removed from an endpoint along with all associated SWID tags.

Supplemental tags are distinct from the other three tag types in that they are used to deploy additional identifying and descriptive information, and to associate that information with any type of tag. As such, they may come into play at any of the lifecycle points discussed above. While supplemental tags are most commonly used to augment information furnished by corpus, primary, or patch tags, they may also be used to augment information contained in other supplemental tags. Supplemental tags are discussed in detail in Section 2.2.4.

2.2.1 Corpus Tags

When products and patches are distributed to a device in preparation for installation, they typically are supplied 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. The SWID specification defines *corpus* tags for vendors and distributors to use to identify and describe products in such a pre-installation state. 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 the package.

Corpus tags may be used by consumers to verify the integrity of an installable product and to 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 (see 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 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) will 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 (see 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.

2.2.2 Primary Tags

Once successfully installed on an endpoint, each tagged product provides at least one tag that, at a minimum, furnishes 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.

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

A globally unique tag identifier is essential information in many usage scenarios because it may be used as a globally unique *proxy identifier*. The tag identifier of a primary tag can be considered a proxy identifier for the tagged product because there is a one-to-one relationship between the primary tag and the software it identifies. 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 primary tag identifiers, rather than their fully populated primary tags.

Because software products may be furnished as suites or bundles or as add-on components for other products, the SWID specification defines a `<Link>` element (see Section 2.4.4) that is used within a SWID tag to document relationships between the product described by the tag and other tagged products that may be available and installed. Three types of relationships are worth noting:

- **Parent.** To document situations where the product described by the primary tag is part of a larger group of installed software, the primary tag points to the primary tag of the larger software group using a `<Link>` element where the `@rel` attribute is set to `parent`.
- **Component.** To document situations where the product described by the primary tag is a component of a separately installable software product, the primary tag points to the primary tag of the product of which it is a component using a `<Link>` element where the `@rel` attribute is set to `component`.

- **Requires.** To document situations where the product described by the primary tag depends on a separately installable software product, the primary tag points to the primary tag of the required product using a `<Link>` element where the `@rel` attribute is set to `requires`.

2.2.3 Patch Tags

A patch tag describes localized changes made to a previously installed 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 are treated as identifying and describing the patch rather than the product to which the patch was 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.

Patch tags will include information linking them with the primary tag of the patched product (see Section 2.4.4 on the `<Link>` element). In this way patch tags may assist in determining whether an installed product has all required patches applied. A patch will likely also include a manifest of the new and/or changed files (see 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 are removed, and new tags are installed.

Patch tags use a `<Link>` element with the `@rel` attribute set to `patches` to point to the primary tag of the patched product. Patches may also relate to other patches using two other relationships as follows:

- **Requires.** To document situations where the patch described by the patch tag requires the prior installation of another patch, the patch tag points to the patch tag of the required patch using a `<Link>` element where the `@rel` attribute is set to `requires`.
- **Supersedes.** To document situations where the patch described by the patch tag entirely replaces another patch (which may or may not have already been installed), the patch tag points to the patch tag of the superseded patch using a `<Link>` element where the `@rel` attribute is set to `supersedes`.

The relationships related to patch tags are illustrated in Figure 1.

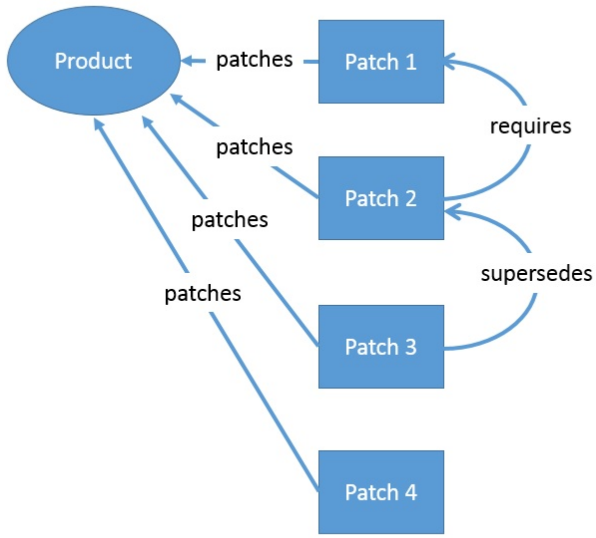

Figure 1: Patch Tag Relationships

In this figure, four patches have been applied over time to a product. Each patch places a patch tag on the device where the patched product resides. Each patch tag includes a `<Link>` element with a `@rel` attribute value of `patches` and a pointer to the patched product's primary tag. Because Patch 1 must be installed before Patch 2 may be installed, the patch tag associated with Patch 2 includes a `<Link>` element with a `@rel` attribute value of `requires` and a pointer to the patch tag for Patch 1. Because Patch 3 entirely replaces Patch 2, the patch tag associated with Patch 3 includes a `<Link>` element with a `@rel` attribute value of `supersedes` and a pointer to the patch tag for Patch 2. Patch 4 is completely independent of the other three patches, so its patch tag does not include any `<Link>` elements pointing to any of the other patch tags.

2.2.4 Supplemental Tags

As noted in the introduction to this section, SWID tags are not supposed to be modified by any entity other than the tag creator. In order to provide a mechanism whereby consumers may add arbitrary post-installation information of local utility, the SWID specification 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.

Any entity may create a supplemental tag, for any purpose. For example, supplemental tags may be created by automated tools in order to augment an existing 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 tagged product's primary tag using a `<Link>` element where the `@rel` attribute is set to *supplemental*. 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.

While not a common usage, supplemental tags may also be employed to augment non-primary tags. For example, a supplemental tag could add local information about a patch tag (e.g., to record a timestamp indicating when the patch was applied), or even about another supplemental tag. In such situations, the supplemental tag also contains a `<Link>` element pointing to the tag that is having its information augmented.

A supplemental tag is intended to furnish data values that 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, if provided by the software producer, is considered the most reliable, and tools can be expected to 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 is the most reliable name.

2.3 Tag Deployment

A SWID tag for a software product could be created on any of these occasions:

- 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 process by a non-authoritative source that obtains a copy of a product after its release to market, and then uses reverse engineering and analysis techniques to create a tag.

Once a tag is created, deployment of that tag to a device could occur in any of three main ways. The first and most common method of tag deployment is for a tag to be incorporated into the product's installation package, which then causes the tag to be installed on an endpoint as part of the software installation procedure. This method is available when the tag creator is in a position to ensure that the tag is included in the installation package.

A second method of tag deployment is to store SWID tags 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 that has been inadvertently deleted,
- To correct a tag that 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 that, 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 are able to 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 in Section 2.2 are distinguished from each other.

A SWID tag (whether corpus, primary, patch, or supplemental) is represented as an XML root element with several sub-elements. `<SoftwareIdentity>` is the root element, and it 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).

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". In the SWID specification, a value for `@version` is **optional** and defaults to "0.0". (Note that later in this document, guidance is provided that **requires** a value for `@version` in corpus and primary tags.)

- `@versionScheme`: a label describing how version information is encoded, e.g., "`multipartnumeric`". In the SWID specification, a value for `@versionScheme` is **optional** and defaults to "`multipartnumeric`". (Note that later in this document, guidance is provided that **requires** a value for `@versionScheme` in corpus and primary tags.)

- `@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 that allows one tag for a software product to supersede another, without suggesting 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).

Under normal conditions, it would be unexpected to discover multiple tags present in the same location on a device that all identify the same installed product, have the same `@tagId`, but have different `@tagVersion` values. Such a situation probably reflects a failure to properly maintain the device's inventory of SWID tags. Nevertheless, should such a situation be encountered, the tag with the highest `@tagVersion` is considered to be the valid tag, and the others may be ignored.

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

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

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

Table 1 illustrates how the tag type may be determined by inspecting the values of `@corpus`, `@patch`, and `@supplemental`. If all these values are false, the tag type is *primary*. Otherwise, at most one of `@corpus`, `@patch`, or `@supplemental` is expected to be true. In Sections 4.3.1 and 4.4.1 of this document, guidelines are provided that require patch and supplemental tags to include a `<Link>` element associating them with the tags to which they are related.

Table 1: How Tag Types Are Indicated

Tag Type	`@supplemental`	`@patch`	`@corpus`	`<Link>` required `@rel`
Corpus	`false`	`false`	`true`	`N/A`
Primary	`false`	`false`	`false`	`N/A`
Patch	`false`	`true`	`false`	`patches`
Supplemental	`true`	`false`	`false`	`supplemental`

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 (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 is “com.acme.rms-sensor-1”. The `<Entity>` element (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 (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 that are not included in the product’s primary tag (e.g., site-specific information such as contact information for the information 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 identifies, at minimum, the organizational or individual entity that 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 (e.g., @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 contains a value for @role, and additionally, every tag contains an <Entity> element identifying the tag creator. Values for @role are selected from an extensible set of allowed tokens, including these:

    - **aggregator:** an entity that packages sets of products and makes them available as single installable items

    - **distributor:** an entity that handles distribution of products developed by others

    - **licensor:** an entity that handles licensing on behalf of others

    - **softwareCreator:** an entity that develops software products

    - **tagCreator:** an entity that creates SWID tags

Values for @regid are URI references as described in RFC 3986 [RFC 3986]. To ensure interoperability and to allow for open source project support, Section 6.1.5.2 of the SWID specification recommends 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 (a string without a URI scheme specified is defined to use the "`http://`" scheme).
- 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`).

For tag creators that do not have a domain name, the `mailto` scheme may be used in place of the `http` scheme to identify the tag creator by email address, e.g., `mailto:foo@bar.com`.

The example below illustrates a SWID tag containing two `<Entity>` elements. The first `<Entity>` element identifies the single organization that is both the software creator and the tag creator, and a second element identifies the organization that 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. If no absolute directory is provided, the directory is considered to be relative to the directory location of the SWID tag.

- `<File>`: files discovered and believed to be part of the product. If no absolute directory path is provided, the file location is assumed to be relative to the location of the SWID tag. If a parent `<Directory>` includes a nested `<File>`, the indicated file is relative to the parent location.
- `<Process>:` related processes discovered on the device.
- `<Resource>`: other general information that may be included as part of the product.

Note that `<Evidence>` is represented in a SWID tag in the same manner as `<Payload>` (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. The href can point to several different values including:
    - a relative URI

  - a physical file location with any system-acceptable URI scheme (e.g., file://, http://, https://, ftp://)
  - a URI with "swid:…" as the scheme, which refers to another SWID tag by tagId.
  - a URI with "swidpath:..." as the scheme, which contains an XPATH query [XPATH 2.0]. This XPATH would need to be resolved in the context of the system by software that can lookup other SWID tags and select the appropriate tag based on the query.

- `@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="patches"
    href="swid:com.acme.rms-ce-v4-1-5-0">
  …
</SoftwareIdentity>
```

In this example, the patch has its own `@tagId` and `@version`, and it 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 the product. Several `<Meta>` attributes of interest are highlighted below:

- `@activationStatus`: identifies the activation status of the product. The SWID specification provides several example values (e.g., `Trial`, `Serialized`, `Licensed`, and `Unlicensed`), but any string value may be supplied. Valid values for `@activationStatus` are expected to be worked out over time by tag implementers.

- `@colloquialVersion`: the informal version of the product (e.g., `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). 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 requires Service Pack 1 or later of a specific product installed on all devices, the organization 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.) that 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="EPV12.cab" size="1024000"
             SHA256:hash="a314fc2dc663ae7a6b6bc6787594057396e
6b3f569cd50fd5ddb4d1bbafd2b6a" />

       <File name="installer.exe" size="524012"
             SHA256:hash="54e6c3f569cd50fd5ddb4d1bbafd2b6ac41
28c2dc663ae7a6b6bc67875940573" />

   </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 recognize such tag modifications, it is necessary to validate that a SWID tag collected during an inventory or discovery process has not had specific elements 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:

> Signatures are not a mandatory part of the software identification 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 references 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.[3] Users may also include a hexadecimal hash string (the "thumbprint") to document the

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

relationship between the tag entity and the signature, using the `<Entity>` `@thumbprint` attribute.

Section 6.1.10 of the SWID 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 shall include a timestamp provided by a trusted timestamp server. This timestamp shall be provided using the XAdES-T form. The SWID tag shall also include the public signature for the signing entity.

Section 6.1.10 of the SWID specification 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 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"
           SHA256:hash="a314fc2dc663ae7a6b6bc6787594057396e6b3f569c
d50fd5ddb4d1bbafd2b6a"/>
        <File name="sensors.dll" size="13295"
           SHA256:hash="54e6c3f569cd50fd5ddb4d1bbafd2b6ac4128c2dc66
3ae7a6b6bc67875940573"/>
      </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 is 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 specification 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 US 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. Guidance items are grouped into the following categories:

- **GEN:** General guidance applicable to all types of SWID tags.
- **COR:** Guidance specific to corpus tags.
- **PRI:** Guidance specific to primary tags.
- **PAT:** Guidance specific to patch tags.
- **SUP:** Guidance specific to supplemental tags.

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 (as defined in Section 2.2). Section 5 provides information on several usage scenarios. Whereas Sections 3 and 4 establish minimum requirements for 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 that conform to all requirements defined in the ISO/IEC 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 to address implementation issues where the SWID specification is silent or ambiguous by adding new “SHOULD” or “MUST” clauses.

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 (or what) 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 first or second party to the creation, maintenance, and distribution of the software. An authoritative tag creator may be a first party if it creates the software, and a second party if it aggregates, distributes, or licenses software on behalf of the software creator. Essentially, any party that is involved in tag creation as part of the process of releasing software is considered an authoritative tag creator. Such parties tend to possess 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) that is in a third-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 through other means such as technical analysis of 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 SoftwareIdentity Elements

This section provides draft guidance on implementation of <SoftwareIdentity> elements intended to clarify details that are not directly addressed in the SWID specification.

The SWID specification defines four tag types (corpus, primary, patch, and supplemental), but provides only three Boolean attributes within the <SoftwareIdentity> element to indicate the type of tag (@corpus, @patch, @supplemental). The SWID specification is silent on whether more than one of the three indicators may be set to true at any one time. Because it does not make sense to set more than one of the tag-type indicators to be true, guidance to that effect is provided here.

GEN-2. At most one of the following <SoftwareIdentity> attributes may be set to true: @corpus, @patch, @supplemental.

3.4 Implementing 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. This is useful information, but does not make clear how a tag consumer might inspect a tag and determine whether the tag was created by an authoritative or non-authoritative entity. This section provides clarifying guidance on this point.

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 a single <Entity> element that specifies only the tag creator role, tag consumers can assume that the tag creator is non-authoritative. To enable tag consumers to accurately determine that a tag is created by an authoritative source, 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.

If this guidance is observed, tag consumers may reliably distinguish authoritative and non-authoritative tag creators according to 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, otherwise the tag creator is non-authoritative. This idea leads to the following guidance:

> **GEN-3.** [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. Second-party authoritative tag creators SHOULD provide one or more additional <Entity> elements or a single <Entity> element with multiple @role attribute values specifying at least one of these predefined roles: “aggregator”, “distributor”, “licensor”.

Non-authoritative tag creators may be unable to accurately determine and identify the various entities associated with a software product, including the software creator. Nevertheless, because tag consumers may obtain substantial benefits from information about each product’s software creator, non-authoritative tag creators are encouraged to include this information in a tag whenever possible.

> **GEN-4.** [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 that created the software product.

3.5 Implementing Payload and Evidence File Data

Files comprising a product or patch are enumerated within `<Payload>` (by authoritative tag creators) or `<Evidence>` (by non-authoritative tag creators) elements using the `<File>` element.

The SWID specification requires only that the `<File>` element specify the name of the file, using the `@name` attribute. This information is insufficient for most cybersecurity usage scenarios. Additional information is needed to enable cybersecurity 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, cybersecurity processes may rapidly and efficiently test for changes that alter a file's size.

> **GEN-5.** 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.

Knowing a file's expected size is useful and enables a quick check to determine whether a file may have changed. Because improper changes may also occur in ways that do not alter file sizes, file hash values are also necessary. If there is a difference in the files' sizes, a change has occurred. If the size is the same, re-computing a hash will be necessary to determine if a change has occurred.

Authoritative tag creators are expected to have sufficient knowledge of product details to be able to routinely provide hash values. Non-authoritative tag creators may not have the necessary knowledge of or access to files to provide hash information, but are encouraged to do so whenever possible.

> **GEN-6.** [Auth] Every `<File>` element within a `<Payload>` element MUST include a hash value.

> **GEN-7.** [Non-Auth] Every `<File>` element within an `<Evidence>` element SHOULD include a hash value.

When selecting a hash function, it is important to consider the support lifecycle of the associated product. The hash value will likely be computed at the time 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 NIST SP 800-57 Part 1 [SP800-57-part-1], when applying a hash function over a time period that extends beyond the year 2031, a minimum security strength of 128 bits is needed. Weak hash values are of little use and should be avoided.

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

Software products tend to be used long beyond the formal product support period. Stability in the hash functions used within SWID tags is 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-9.** [Auth] Whenever a `<Payload>` element is included in a tag, every `<File>` element contained therein MUST provide a hash value based on the SHA-256 hash function.

> **GEN-10.** [Non-Auth] Whenever an `<Evidence>` element is included in a tag, every `<File>` element contained therein SHOULD provide a hash value based on the SHA-256 hash function.

> **GEN-11.** 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.

Due to the use of 64-bit word values in the algorithm, use of SHA-512 hash function implementations may perform better on 64-bit systems. For this reason, tag creators are encouraged to consider including a SHA-512 hash value, since this might provide for a better performing integrity measure.

3.6 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. It points out that:

> To prove authenticity of a software identification tag, for example to validate that the software identification tag collected during a discovery process has not had specific elements of the tag altered, authentication is supported through the use of digital signatures within the software identification tag.

Information gathered through the examination of SWID tags is used to support automated and human decision making. As a result, it is important to be able to authenticate and measure the integrity of a SWID tag,

GEN-12. Use of XML digital signatures is RECOMMENDED.

This section provides additional guidance to provide a reproducible, interoperable, and verifiable framework for generation and use of XML 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 implementations handle default values when digitally signing tags? Consider requiring values for all attributes with no assumption of a default value.

3.7 Referring to Product Installation Packages, Releases, and Patches

The SWID specification requires that every tag include a globally unique identifier, called the *tag identifier* (or tag ID), recorded in the `<SoftwareIdentity>` `@tagId` attribute. The tag ID is a particularly critical piece of information, because it may be used by other asset management or cybersecurity processes as a software identifier. This section elaborates that idea and provides guidance on how tag identifiers may be used to refer to product installation packages, product releases, and product patches.

As discussed in Section 2.2.1, corpus tags identify and describe software products in a pre-installation state. Organizations may find it useful to be able to refer to such pre-installation versions of products, for example, to enumerate lists of products approved for installation within an enterprise. Thus the tag identifier of a corpus tag should be considered a valid and reliable identifier of pre-installation products, leading to the following guidance.

GEN-13. The `@tagId` of a corpus tag MAY be used in any system, document, or process to designate a software product in its pre-installation state.

Similarly, because primary tags identify and describe software products that are installed on endpoints, organizations may find it useful to be able to refer to installed versions of products using the tag identifiers of those products' primary tags.

GEN-14. The `@tagId` of a primary tag MAY be used in any system, document, or process to designate a software product in its post-installation state.

Lastly, because patch tags identify and describe patches that have been applied to released software products, organizations may find it useful to be able to refer to patches using the tag identifiers of patch tags.

> **GEN-15.** The `@tagId` of a patch tag MAY be used in any system, document, or process to designate a software patch.

Because supplemental tags are used to add information to corpus, primary, and patch tags, their tag identifiers are only useful in situations where there is a need to refer specifically to the supplemental tag itself. Tag identifiers of supplemental tags should not be used as proxy identifiers for software installation packages, installed software products, or software patches.

> **GEN-16.** The `@tagId` of a supplemental tag SHOULD NOT be used in any system, document, or process to designate a pre-installation software package, an installed software product, or a software patch.

Tag identifiers are comparable to International Standard Book Numbers (ISBNs) for books. When the descriptive metadata about a book is revised or extended (in, say, a database containing records describing books for sale), the book itself does not change, and so its ISBN does not change. A SWID tag is like a record in a bookseller's database, containing identifying and descriptive metadata about a pre-installation software package, an installed software product, or a software patch. When the metadata is revised or extended, but there is no associated change to the installation package, product, or patch, the tag identifier should not change.

3.8 Updating Tags

Although the SWID specification does not prohibit modification of SWID tags, it does restrict modifications so that they can only be performed by the original tag creator. The primary reason for altering a tag after it has been installed on a device is to correct errors in the tag. In rare circumstances it may be useful to update a tag to add data elements that logically belong in the tag and not in a separate supplemental tag. But under normal conditions, tags should rarely be modified, and supplemental tags should be used to add identifying and descriptive product information.

When changes are made to a product's codebase that cause the product's version to change, those changes should be reflected by removing all original tags (primary, supplemental, and patch tags) and installing new tags as appropriate to identify and describe the new product version. Patches should be indicated by adding a patch tag to the installed collection of tags.

When an existing tag must be updated, it will rarely make sense to edit the tag in place, that is, to selectively modify portions of the tag as if using a text editor. Such editing actions would likely invalidate XML digital signatures stored in the tag. Thus it is expected that when a tag is updated, it is always fully replaced, and any stored digital signatures are replaced as well.

When a tag must be updated to correct errors or add data elements, its `<SoftwareIdentity>` `@tagId` should not be changed. This is because, as discussed in Section 3.7, tag identifiers may be used as identifiers for pre-installation software packages, installed software products, or software patches. It is important that tag identifiers be usable as reliable persistent identifiers. This leads to the following guidance.

> **GEN-17.** When it is necessary to update a tag to correct errors in or add data elements to that tag, the tag's `<SoftwareIdentity>` `@tagId` SHOULD NOT be changed.

When tags are updated, however, it is important that the updates be implemented in a manner that supports easy change detection. Tag consumers should not be required or expected to fully process all discoverable tags on an endpoint in order to determine whether any of the products have changed since the last time the tags were examined. To enable easy change detection, tag creators are required to update the <SoftwareIdentity> @tagVersion attribute to indicate that a change has been made to the tag.

> **GEN-18.** When it is necessary to update a tag to correct errors in or add data elements to that tag, the tag's <SoftwareIdentity> @tagVersion attribute MUST be changed.

If this guidance is observed, tag consumers need only to maintain records of tag identifiers and tag versions discovered on endpoints. If a tag with a previously unseen tag identifier is found on an endpoint, a tag consumer may conclude that a new product has been installed since the last time the endpoint was inventoried. If a tag with a previously discovered tag identifier can no longer be discovered on an endpoint, a tag consumer may conclude that a software product has been removed since the last time the endpoint was inventoried. If, however, a tag is discovered on an endpoint with a previously seen tag identifier but a new tag version, a tag consumer may conclude that identifying or descriptive metadata in that tag has been changed, and so the tag should be fully processed.

3.9 Questions for Feedback

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

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

3.10 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 (as defined in 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 (as defined in Section 2.2): *corpus* tags (Section 4.1), *primary* tags (Section 4.2), *patch* tags (Section 4.3), and *supplemental* tags (Section 4.4).

4.1 Implementing Corpus Tags

As noted earlier (in Section 2.2.1), a corpus tag is a tag where the value of the <SoftwareIdentity> @corpus attribute is set to "true". This section provides guidance addressing the following topics related to implementation of corpus tags: specifying @version and @versionScheme (Section 4.1.1), specifying Payload information (Section 4.1.2), and signing corpus tags (Section 4.1.3).

4.1.1 Specifying the Version and Version Scheme in Corpus Tags

Corpus tags identify and describe software products in a pre-installation state. As part of the process of determining whether a given product is suitable for or allowed to be installed on an endpoint, tag consumers often need to know the product's specific version. The SWID specification provides the <SoftwareIdentity> @version attribute for recording version information, but defines this attribute as optional and defaulting to a value of "0.0".

This document seeks to encourage software providers both to assign and maintain product versions for their products, and to explicitly record those versions in appropriate tags released along with those products. In short, if a software product has an assigned version, that version must be specified in the tag.

> **COR-1.** If a software product has been assigned a version by the software provider, that version MUST be specified in the <SoftwareIdentity> @version attribute of the product's corpus tag, if any.

For many cybersecurity purposes, it is important to know not only a product's version, but also to know whether a given product version represents an "earlier" or "later" release of a product, compared to a known version. For example, security bulletins often warn that a newly-discovered vulnerability was found in a particular version V of a product, but may also be present in "earlier versions." Thus, given two product versions V1 and V2, it is important to be able to tell whether V1 is "earlier" or "later" than V2.

In order to make such an ordering decision reliably, it is necessary to understand the structure of versions and how order is encoded in versions. This is no single agreed-upon practice within the software industry for versioning products in a manner that makes clear how one version of a product relates to another. The "Semantic Versioning Specification" [SEMVER] is one example of a grass-roots effort to recommend a common interpretation of multi-part numeric versions, but it is by no means universal.

The SWID specification defines the <SoftwareIdentity> @versionScheme attribute to record a token that designates the "scheme" according to which the value of <SoftwareIdentity> @version can be parsed and interpreted. Like @version, the

SWID specification defines `@versionScheme` as “optional” with a default value of `multipartnumeric`. But the specification does not define the semantics of the `multipartnumeric` scheme, nor does it explain how additional schemes will be defined and given semantics.

It is beyond the scope of this document to fully resolve those matters. Instead, the following guidance is provided in consideration of the fact that tag consumers have a critical interest in knowing not only a product’s version, but also its versioning scheme and the semantics of that scheme.

> **COR-2.** If a corpus tag contains a value for the `<SoftwareIdentity>` `@version` attribute, it MUST also contain a value for the `<SoftwareIdentity>` `@versionScheme` attribute.

> **COR-3.** Whenever a value for the `<SoftwareIdentity>` `@versionScheme` attribute is provided in a corpus tag, it MUST be selected from a well-known public list of version scheme identifiers. Such public lists SHOULD specify semantics for each version scheme sufficient for comparing two versions and determining their relative order in a sequence.

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

4.1.3 Corpus Tag Signing

Corpus tags are helpful when performing product authenticity and integrity checks. For this to work, the tags themselves must be digitally signed to ensure that the data values contained within them, including the `<Payload>` details, have not been modified, and a separate signature is required to support authentication of the provider of each tag.

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

4.2 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 cybersecurity processes on information systems, additional requirements must be satisfied. This section provides guidance addressing the following topics: specifying version and version scheme information (Section 4.2.1), specifying `<Payload>` or `<Evidence>` information (Section 4.2.2), and specifying attributes needed to form CPE names (Section 4.2.3).

4.2.1 Specifying the Version and Version Scheme in Primary Tags

Primary tags identify and describe software products in a post-installation state. Like corpus tags, primary tag information about product versions and associated version schemes is important to enable tag consumers to conduct various cybersecurity operations. Unlike the case for corpus tags, however, guidance for primary tags must distinguish between authoritative and non-authoritative primary tag creators.

> **PRI-1.** [Auth] If a software product has been assigned a version by the software provider, that version MUST be specified in the `<SoftwareIdentity>` `@version` attribute of the product's primary tag.

> **PRI-2.** [Auth] If a primary tag contains a value for the `<SoftwareIdentity>` `@version` attribute, it MUST also contain a value for the `<SoftwareIdentity>` `@versionScheme` attribute.

> **PRI-3.** [Non-Auth] If a software product has been assigned a version by the software provider, that version SHOULD be specified in the `<SoftwareIdentity>` `@version` attribute of the product's primary tag if it can be determined.

> **PRI-4.** [Non-Auth] If a primary tag contains a value for the `<SoftwareIdentity>` `@version` attribute, it SHOULD also contain a value for the `<SoftwareIdentity>` `@versionScheme` attribute if an accurate version scheme can be determined.

> **PRI-5.** Whenever a value for the `<SoftwareIdentity>` `@versionScheme` attribute is provided in a primary tag, it MUST be selected from a well-known public list of version scheme identifiers. Such public lists SHOULD specify semantics for each version scheme sufficient for comparing two versions and determining their relative order in a sequence.

4.2.2 Primary Tag Payload and Evidence

Detailed information about the files comprising an installed software product is a critical need for cybersecurity operations. 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-6.** [Auth] A `<Payload>` element MUST be provided, in either a software product's primary tag or a supplemental tag.

> **PRI-7.** [Non-Auth] An `<Evidence>` element SHOULD be provided, in either a software product's primary tag or 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-8. `<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-9. [Auth] The `<Payload>` element MUST list every machine instruction file comprising the product described by the tag.

PRI-10. [Non-Auth] The `<Evidence>` element MUST list every machine instruction file comprising the product described by the tag.

4.2.3 Specifying Attributes Required to Form CPE Names

A component of NIST's Security Content Automation Protocol (SCAP), CPE is a standardized method of naming classes of applications, operating systems, and hardware devices present among an enterprise's computing assets.[4] NIST maintains a dictionary of CPE names as part of the National Vulnerability Database (NVD).[5] 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.

The SWID specification defines several `<Meta>` element attributes that are needed to support CPE name generation. These attributes are:

- `@product`: This attribute provides the base name of the product (e.g., Acrobat, Creative Suite, Office, Websphere, Windows). 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.

---

4 See: http://scap.nist.gov/specifications/cpe/.

5 See: https://nvd.nist.gov/.

- `@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).

If these attributes are specified, a valid CPE name can be mechanically generated. Appendix A describes an algorithm that may be used to generate such CPE names.

Guidance is as follows:

> **PRI-11.** A `<Meta>` element MUST be included in a product's primary tag. If appropriate values exist and can be determined, the `<Meta>` element MUST furnish values for the following attributes: `@product`, `@colloquialVersion`, `@revision`, and `@edition`.

4.3 Implementing Patch Tags

As noted earlier in Section 2.2, a patch tag has the value of the `<SoftwareIdentity>` `@patch` attribute set to `true`. This section provides guidance addressing the following topics related to implementation of patch tags: linking patch tags to related tags (Section 4.3.1) and specifying `<Payload>` or `<Evidence>` information (Section 4.3.2).

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.** 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:
>
> - **If 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.
>
> - **If 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 primary tag of the affected product, with swidpath: as its scheme, containing an XPATH query that can be resolved in the context of the system by software that can look up 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. The guidance below is limited to authoritative tag creators, since it cannot be assured that non-authoritative creators of patch tags will be able to provide the necessary information.

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:

- **If 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.

- **If 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:

- **If 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.

- **If 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 that 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 changes:

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. As stated before in requirements GEN-5 and GEN-6, 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 that 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 pre-existing file has been modified on the device
- The string value “remove” to indicate a pre-existing 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 that enumerates every file that was used as part of the detection process.

4.4 Implementing Supplemental Tags

A supplemental tag has the value of the <SoftwareIdentity> @supplemental attribute set to true. This section provides guidance addressing the following topics related to implementation of supplemental tags: linking supplemental tags to other tags (Section 4.4.1), and the precedence of information contained in a supplemental tag (Section 4.4.2).

4.4.1 Linking Supplemental Tags to Other Tags

An individual supplemental tag may be used to furnish data elements that complement or extend data elements furnished in another individual tag. That is, a supplemental tag may not be used to supplement a collection of tags. A supplemental tag may supplement any type of tag, including supplemental tags. Because the SWID specification does not clearly state how a supplemental tag should indicate its linkage to other tags, clarifying guidance is provided here.

> **SUP-1.** A supplemental tag MUST contain a <Link> element to associate itself with the individual tag that it supplements. The @rel attribute of this <Link> element MUST be set to supplemental. The @href attribute MUST be set to a URI with swid: as its scheme, followed by the @tagId of the tag that is being supplemented.

4.4.2 Precedence of Information

As noted earlier, a supplemental tag is intended to furnish data elements that complement or extend data elements furnished in another tag. This does not preclude situations in which a supplemental tag contains elements or attributes that potentially conflict with elements or attributes furnished in the tag being supplemented. For example, suppose an endpoint contains a primary tag where the value of the <SoftwareIdentity> @name attribute is specified to be Foo, and a supplemental tag is also present that is linked to the primary tag but specifies the value of the <SoftwareIdentity> @name attribute to be Bar.

One option is to treat any conflicting data items in a supplemental tag as overriding the corresponding values provided in the tag that is supplemented. Choosing this treatment, however, would introduce a new complexity, since multiple supplemental tags could all point to the same supplemented tag, and all data values could conflict. The only way to resolve this would be to add new requirements to establish precedence orders among supplemental tags.

Instead, this document takes the position that supplemental tags strictly extend, and never override. So in the example above, Foo is considered to be the correct value for @name, and the value of Bar furnished in the supplemental tag is ignored.

Because certain attribute values pertain to tags themselves—e.g., @tagId, @tagVersion, and Entity information about the tag creator—differences in those values between a supplemental tag and a supplemented tag are never construed as conflicts. In other cases, information in a supplemental tag may be *combined* with information in the supplemented tag to obtain a full description of the product. For example, a primary tag may provide an <Entity> element that specifies the tagCreator role, while a supplemental tag provides <Entity> elements specifying other roles such as softwareCreator and licensor. In this scenario, the primary and supplemental tag collectively furnish all Entity roles. If, however, both the primary and supplemental tags provide `<Entity>` elements specifying values for the same role (e.g., both tags specify different `softwareCreator` values), then the conflicting value in the supplemental tag is ignored.

This leads to the following guidance.

> **SUP-2.** If a supplemental tag provides a data value that conflicts with corresponding data values in the supplemented tag, the data value in the supplemented tag MUST be considered to be the correct value.

4.5 Summary

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

- Corpus tags must include `<Payload>` details, and must be digitally signed to facilitate authentication and integrity checks.
- Authoritative creators of primary tags are required to provide `<Payload>` information, and to include `<Meta>` attribute values needed to support automated generation of CPE names. Non-authoritative creators of primary tags are required to provide `<Evidence>` information for any data used to detect the presence of the product.
- 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.
- Supplemental tags may supplement any type of tag, but must be explicitly linked to the supplemented tag. Any data value supplied in a supplemental tag that conflicts with a corresponding data value in the supplemented tag is ignored.

5 SWID Tag Usage Scenarios

Proper identification and management of the software deployed on an organization's endpoints enables security professionals to manage a number of security and operational risks. Through the application of SAM practices, organizations can 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 requirements in the previous sections provide for interoperability in the creation of SWID tags that may be used by SAM and configuration management products to provide situational awareness. For example, these products can evaluate the difference between the observed SWID tag-based software inventory and a desired state specification defined by the organization through digital policies. Continuous monitoring processes can use the SWID tag data to identify and report unexpected changes, 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.

This section describes a number of usage scenarios for SAM activities, organized into a set of steps as described in

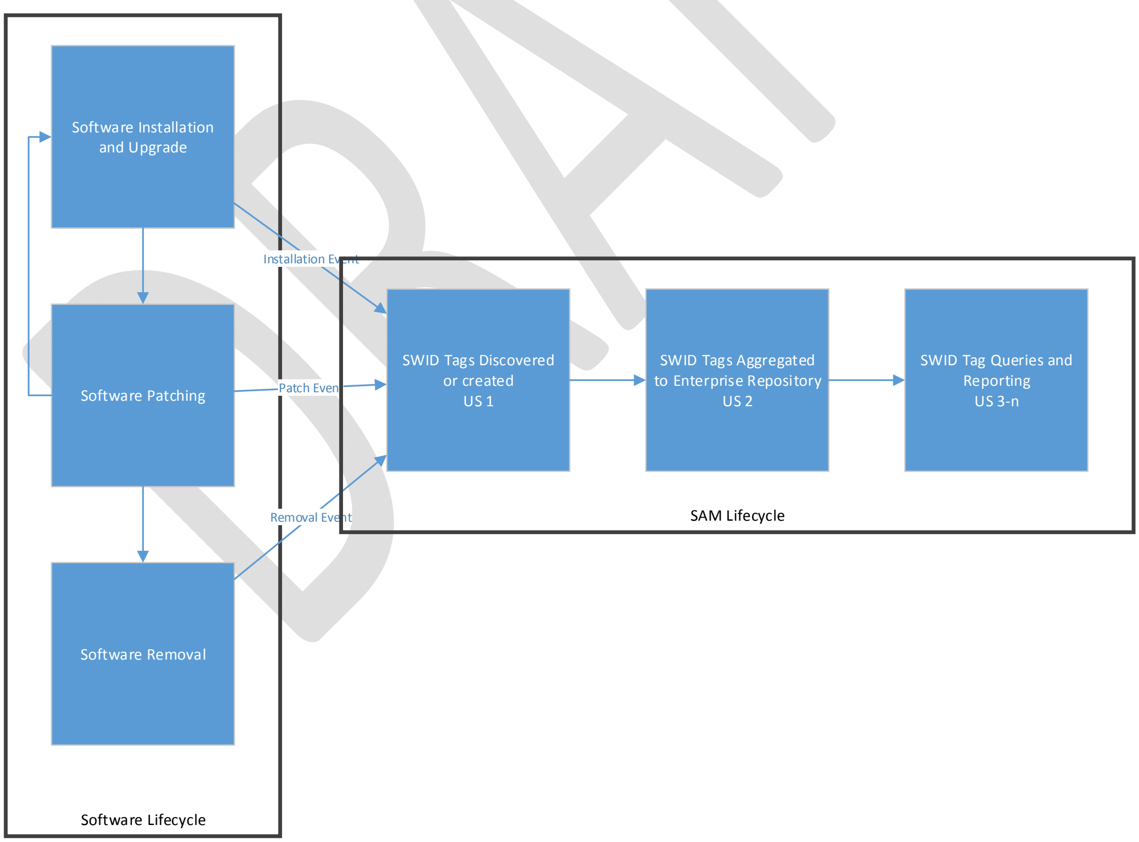

Figure 2.

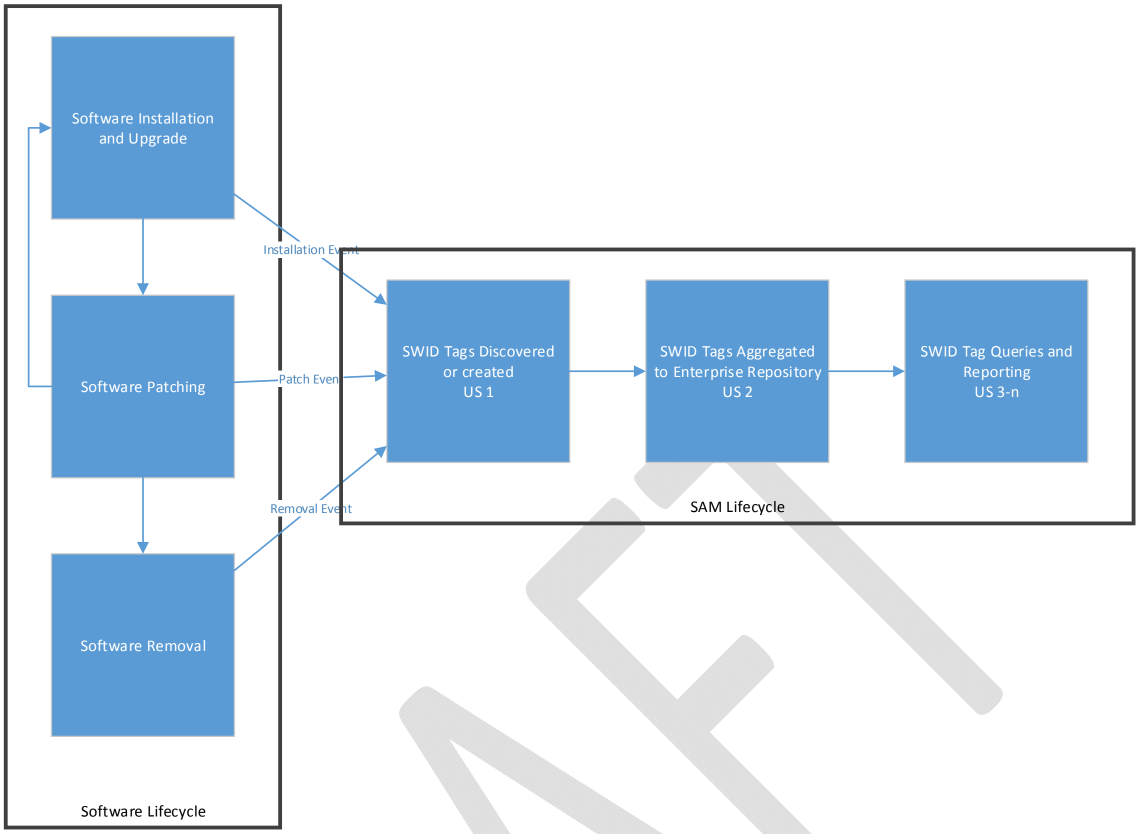

Figure 2: SWID Tag-Related SAM Process Steps

These steps include discovering, collecting, searching, using, and reporting software inventory. These usage scenarios are not intended to represent an exhaustive list of possible SWID applications. They are intended to provide informative examples regarding the use of the SWID specification to accomplish organizational needs.

Each usage scenario in the following sections describes specific process steps that illustrate the use of SWID tags to accomplish each scenario. Each process is accompanied by assumptions, if any, that must be true to complete those steps and achieve the expected outcomes of that process.

5.1 Software Inventory Management

To properly manage software it is first necessary to know what software is deployed to endpoints within an organization's enterprise environment. The process of gathering this knowledge can be broken down into two separate but related software inventory functions: collection and reporting. These functions are described in greater detail in usage scenarios 1 and 2. Together these functions provide the data that is needed to support various search and analysis capabilities that provide the knowledge necessary to support operational decision making. An example of this knowledge is determining if a software product is authorized for use, meets licensing requirements, and is properly patched against vulnerabilities.

Software inventory may be maintained in both local and remote repositories. For example, management of a local repository enables SWID tag information usage within an endpoint, such as to perform execution authorization or file integrity checks quickly. A local inventory is also useful when network connectivity is not immediately available. Aggregating software inventory information to a downstream component allows for a larger context (e.g., multiple endpoints to be considered) and for integration with other enterprise capabilities (e.g., continuous monitoring). Remote repositories may themselves report software inventory to other downstream repositories, such as to an enterprise repository, as pictured in Figure 3.

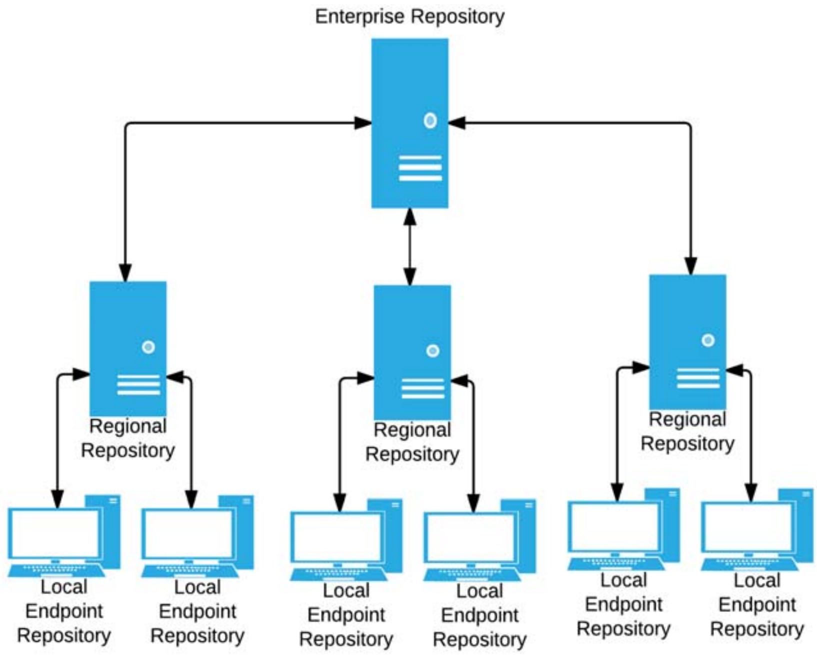

Figure 3: Conceptual Hierarchy of Software Inventory Repositories

5.1.1 Usage Scenario 1: Discovering and Collecting Software Inventory Information within an Endpoint

The first step to managing software on an endpoint is to know what software is installed on it – its software inventory. By monitoring software change events and using information provided by SWID tags, maintaining an up-to-date inventory is possible. This enables a current and accurate understanding of what software is installed on an endpoint. As software changes are made, the endpoint's software inventory must be updated to reflect those changes. Modifications occur throughout the software lifecycle, as shown in Figure 2, including:

- Installing software
- Upgrading software
- Patching software
- Removing software
- Modifying or removing a SWID tag

One or more software discovery products can analyze an endpoint for software changes, either on an event-driven basis or through periodic assessment of installation locations. These changes include detecting and processing modifications to existing SWID tags on the endpoint. It is important to note that this analysis should consider various sources for performing this discovery, including:

- The endpoint's local, directly attached filesystems, including files installed by traditional installation utilities and archived distributions (e.g., tar, zip)
- Temporary storage connected to the endpoint (e.g., external hard drive, USB device)
- Software contained in native package installers (e.g., RPM Package Manager (RPM))
- Shared filesystems (e.g., a mapped network drive or network attached storage) that contain software that is executable from an endpoint

As illustrated in Figure 4, a primary use of SWID tags is to identify and provide information about an endpoint's installed software. For tagged software, authoritative SWID tags on the endpoint can be used to identify installed software. As discussed in Section 3.2, for untagged software, software discovery products can also place non-authoritative SWID tags on the endpoint. This is an important capability, since it is likely that some software will be untagged at the point of installation.

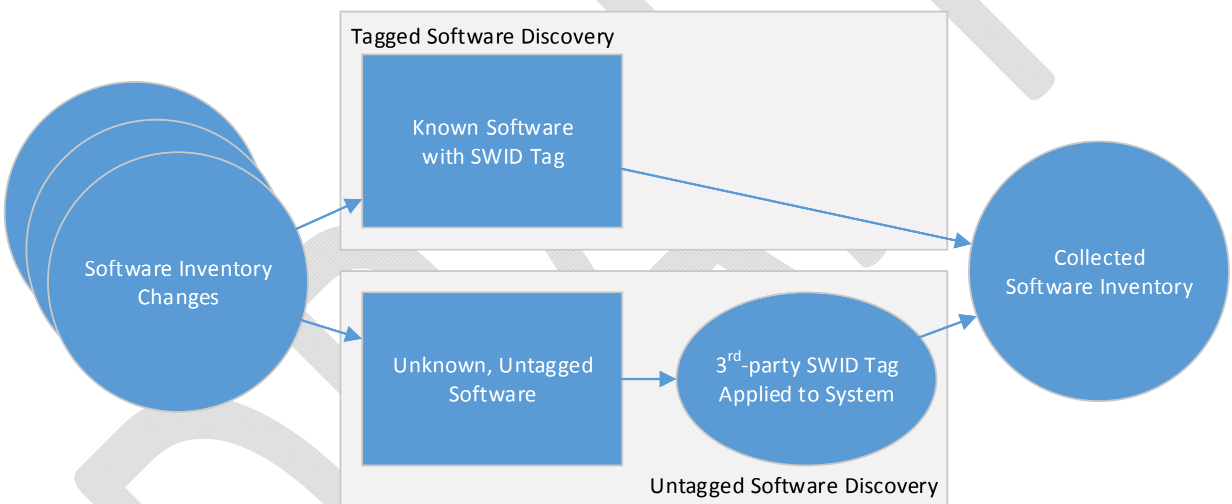

Figure 4: Notional Software Discovery Patterns

5.1.1.1 Assumptions

This usage scenario assumes the following conditions:

- The discovery tool has sufficient access rights to the endpoint to discover each software instance and any metadata related to the software instance. This includes access rights to read SWID tag information on the endpoint.
- Not all installed software will have an associated SWID tag.

5.1.1.2 Process

During the analysis process the following steps are performed for each identified software change:

1. Upon detecting new or changed software in an installation location or a mounted filesystem on the endpoint, the SAM tool will attempt to discover appropriate SWID tags in that location.

2. The SAM tool will update the local endpoint repository with the data from the existing SWID tags, creating entries for software products and their components. The updates may include:

    - New software items (or subcomponents) that were not previously in the inventory;
    - Changes or updates to existing software products;
    - Files changed, removed, or added by a software patch;
    - Software products removed that were previously included in the inventory; or,
    - New or modified SWID tags, as indicated by new `@tagId` attribute or the same tag id, but new `@tagVersion` attribute of the `<SoftwareIdentity>` element.

3. If a tag was not installed with the software, the SAM tool will create a non-authoritative tag on the endpoint for each instance of an application discovered. The tag will include relevant data about what information was used to discover the installed software products using the `<Evidence>` element.

4. The local repository will be updated, including notification to applicable reporting systems in the enterprise. The repository will track the changes discovered to support SAM and security needs. This includes the location of discovered tags, to enable subsequent extraction of the information contained in each tag, when needed.

5.1.1.3 Outcomes

This combination of activities provides a standardized means for identifying and collecting information related to an endpoint's installed software. When used in this way, SWID tags enable the collection of a comprehensive inventory of installed software products by examining the system for installed SWID tags regardless of how the software is delivered to and installed within the endpoint.

5.1.2 Usage Scenario 2: Aggregating Endpoint Software Inventory

As data is collected, as described in Section 5.1.1, SWID tags enable many reporting capabilities for enterprise system software inventories. SWID tags enable accurate and reliable reporting of the software products installed on an organization's endpoints and support the exchange of normalized data pertaining to these products. Together, this information is critical in effectively managing IT 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).

A significant value of SWID tags is their portability across different device types and platforms. SWID tag information may be aggregated and collated from numerous endpoints into intermediate and enterprise repositories, as illustrated in Figure 3. This data may be updated periodically (e.g., every 72 hours) or based on change events, with the latter providing more up-to-date information.

5.1.2.1 Assumptions

This usage scenario assumes the following conditions:

- Each endpoint is maintaining its own software inventory (see section 5.1.1).
- SWID tags to be aggregated from local repositories have been created in accordance with the requirements described in Sections 3 and 4.
- The SAM tool has network connectivity to the endpoints for which software inventory information is to be aggregated.

5.1.2.2 Process

Periodically, the complete set of tags from each endpoint is either sent to the enterprise repository or collected via a remote management interface by the SAM tool, to create a baseline software inventory. Once this baseline inventory has been established, only software changes since the last exchange need to be provided. This provides for efficiencies in the velocity and volume of information that needs to be exchanged.

1. For a given endpoint, the SAM Tool iterates through each tag in the repository, including non-authoritative SWID tags.
2. The endpoint-collected tags are added to the enterprise repository, recording relevant endpoint identification information (host name, IP addresses, etc.), the date/time of the data collection, and data about the discovery tool or remote management interface used.

5.1.2.3 Outcomes

This application of SWID tags enables the organization to use automation for the accurate and timely collection of software inventory information at an enterprise scale. While many of these processes are achievable without SWID tags, the consistent and precise information provided by SWID tags is beneficial for maintaining an accurate and complete enterprise inventory of all software products deployed to endpoints, regardless of the software and platforms used.

5.2 Using SWID Tags

SWID tags contribute to an accurate and reliable SAM inventory that supports searching for specific product information or software characteristics (e.g., prohibited or required software, specific software versions or ranges, software from a specific vendor). The SWID Tag specification provides a rich set of data that may be used with specific query parameters to search for instances of installed software. In addition to the common name and version values, many SWID tags store extended information such as that identified through the `<Link>` and `<Meta>` elements. Details regarding attributes and values that can be useful for queries are described in Section 2.4.

In many cases, the ability to consistently and accurately search for instances of installed software is important to achieving the organization's cybersecurity situational awareness goals. Query results may be used to trigger alerts based on pre-determined conditions (e.g., prohibited software detected) that may be useful in a continuous monitoring context.

5.2.1 Usage Scenario 3: Identifying Instances of an Installed Product or Patch

One common enterprise need is to determine which endpoints have a specific product and/or patch installed. For example, this can be used to confirm that required software or patches are installed.

5.2.1.1 Assumptions

- This usage scenario assumes the existence of a local repository, populated with collected SWID tags that are created in accordance with the requirements described in Sections 3 and 4.
- The list of mandatory software and patches exists.

5.2.1.2 Process

1. The SAM Tool queries the representation of installed tags in the repository, including data from non-authoritative SWID tags, based on the given query parameters. Where a match is identified, the SAM Tool reports the corresponding endpoint identifier and notes relevant version information from the repository in the query results.
2. Where a patch has been recorded as being successfully installed, the SAM Tool can take advantage of relationships to other patches, as described in Section 2.2.3. For example, if the new patch supersedes a former patch, the SAM Tool should take note that the former patch may no longer apply.
3. Similarly, when the SAM Tool locates a patch tag that indicates that a predecessor patch is required (as described in Section 2.2.3), the SAM Tool can use the location and relationship information in the SWID Tag to confirm the presence of the required predecessor tag.
4. The search results are provided through the SAM Tool's dashboard and/or reporting process.

5.2.1.3 Outcomes

The user is able to find instances of the given product or patch on endpoints for which SWID tags have been collected.

5.2.2 Usage Scenario 4: Identifying Endpoints That Are Missing a Product or Patch

Another common need is to determine which endpoints are missing a software product, such as an organizationally required antivirus application. The list of required software may vary by platform (e.g., Windows clients might have one list, Mac clients another, and Linux servers yet another.) Similarly, endpoints that are dedicated to a particular role (e.g., "messaging server") might have unique required or prohibited software lists.

Similarly, consumers often need reports, for security awareness or management purposes, about endpoints that are missing a patch. SAM tools may also need to determine what patches are installed in order to perform a necessary update. While many reports may be performed from the enterprise repository, the endpoint patch update can directly read the inventory of patch tags from the local endpoint repository to enable timely decision support.

5.2.2.1 Assumptions

This usage scenario assumes the existence of an enterprise repository, populated with collected SWID tags that are created in accordance with the requirements described in Sections 3 and 4.

5.2.2.2 Process

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

2. When the SAM Tool locates a patch SWID tag that indicates that another patch is required (as described in Section 2.2.3), the SAM Tool can use the location and relationship information in the SWID tag to check whether the required patch is missing.

3. Where a patch is marked as missing, the SAM Tool can take advantage of relationships to other patches, as described in Section 2.2.3, to see if that missing patch has been superseded by a newer patch. In this case, the SAM Tool can note that the former patch may no longer apply.

4. The SAM Tool searches the SWID tags discovered, confirming that the required tag contents are identified with the necessary endpoints. Where a match is not located, the SAM Tool records the identifier for each endpoint that does not comply with the software installation requirement. For example, if each endpoint is expected to contain an updated antivirus product, the query may return the hostname of each endpoint where no SWID tag associated with that product was located. Optionally, where a match is located, the SAM Tool records the endpoint’s compliant state.

5. The search results are provided through the SAM Tool’s dashboard and/or reporting process.

5.2.2.3 Outcomes

The SAM tool user is able to accurately and quickly identify instances where a required patch or product (or specific version of a required product) is not installed on a given endpoint. The user is able to determine which endpoints meet (or do not meet) specific requirements. This understanding aids in security situational awareness and supports ongoing vulnerability management that may be a part of a continuous monitoring solution.

5.2.3 Usage Scenario 5: Identifying Orphaned Software, Shared Components, and 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. If the orphaned components contain an exploitable flaw, their presence introduces additional security risk. Additionally, SWID tag reporting can identify endpoints that contain items such as binaries and runtime libraries that do not belong to an installed package.

As SWID tag usage becomes commonplace, software providers are encouraged to document the relationships and dependencies among software products, libraries, and other components through the use of authoritative SWID tags. Use of the `<Link>` element as described in Section 2.2.2 enables understanding of how software components relate, supporting software asset management decisions.

5.2.3.1 Assumptions

This usage scenario assumes the following conditions:

- A SWID tag repository is populated with collected SWID tags that are created in accordance with the requirements described in Sections 3 and 4.
- Those SWID tags include pointers to additional SWID tags using the `<Link>` element and the `@rel` and `@href` attributes that are needed to describe a potential child/parent relationship among software products. This use of the `<Link>` element is described in Section 2.2.2.

5.2.3.2 Process

1. For a given endpoint (or set of endpoints), the SAM Tool iterates through each tag in the repository, including non-authoritative SWID tags. The Tool specifically inspects tags indicating relationships to other products as indicated by the `<Link>` element, `@rel` attribute with a value of `"parent"`. Such tags will include an `@href` pointer to the parent software component.
2. For each tag located, the SAM Tool verifies the installation of the parent software by checking for the referenced installation SWID tag.
3. Where a match is <u>not</u> 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.

5.2.3.3 Outcomes

The user is able to identify components of previously installed software products, including patches that were applied but left behind when a product was uninstalled. Using this information, resources can be optimized and security risks can be mitigated.

5.2.4 Usage Scenario 6: Preventing Installation of Prohibited Software

To strictly control what software may or may not be installed on information systems, SAM tools, supported by corpus tag information, can ensure that all installed software on a given endpoint matches the specification of an authorized software baseline, or whitelist, or does not match the specification of a prohibited software list, or blacklist. There might be multiple lists of authorized or unauthorized software. For example, Windows clients might have a list, Mac clients another, and Linux servers another. Similarly, endpoints that are dedicated to a particular role (e.g., "messaging server") might have unique required or prohibited software lists.

As described in Section 2.2.1, corpus tags may be used to authenticate the issuer of the installation before carrying out the installation procedure. Identification information and other data in a corpus tag can be used to authorize or prohibit software installation during the installation procedure. Additionally, if a manifest of the installation files is included in the corpus tag (see Section 2.4.6 on the `<Payload>` element), the installation routine can confirm (from the whitelist) that the software's integrity has been preserved, preventing tampering in software distributions.

5.2.4.1 Assumptions

This usage scenario assumes that the following conditions exist:

- There is at least one whitelist or at least one blacklist.
- A software product/package to be installed includes a corpus tag describing what will be installed.
- If the issuer of the installation is to be verified, the SWID tag must be signed.

5.2.4.2 Process

1. Upon execution of a software installation or update tool, the tool discovers an associated corpus tag included with the software distribution.
2. The installation tool validates the signer's certificate and validates the tag's signature if the corpus tag is signed.
3. The installation tool identifies if the `@tagId` or metadata matches an item in either a whitelist or a blacklist.
4. The installation tool verifies the hashes of the installation files using the `<Payload>` data included in the corpus tag.
5. If the software to be installed is not authorized or is specifically prohibited, if the signer cannot be verified, if the tag's signature is invalid, or if distribution files have been changed, the installation tool discontinues installation of the software product.

5.2.4.3 Outcomes

For the process described above, the application of SWID tags enables the organization to use automation to control installation of software and patches, and to verify the signer and integrity of each installation package prior to installation.

5.2.5 Usage Scenario 7: Detecting Installed Instances of Prohibited Software

As described in Usage Scenario 6, SWID tag information can be used to restrict installation of prohibited software. In some cases preventing installation of such software is impractical. In these cases SWID tags can help a SAM Tool detect unauthorized software after installation. The SAM tool can compare values in the local endpoint repository to software identified in whitelists or blacklists, and take appropriate action.

Using the process in this scenario, a SAM Tool can simply report the detection of such software, trigger an alarm to the software's presence, or even directly prevent execution of the software.

5.2.5.1 Assumptions

This usage scenario assumes the following conditions:

- A SWID tag repository is populated with collected SWID tags that are created in accordance with the requirements described in Sections 3 and 4.
- There is at least one whitelist or at least one blacklist.

5.2.5.2 Process

1. Through a dashboard or other internal process, the SAM tool is provided with a set of SWID tags that represent a whitelist or a blacklist.
2. The SAM Tool iterates through the recorded tags in the repository, including non-authoritative SWID tags, associated with one or more endpoints on which to report.
3. The SAM Tool parses the values contained in the @name and @version attributes of the <SoftwareIdentity> element. The tool compares each value to the list provided in step 1.
4. If additional confirmation is required, such as to help prevent 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.
5. Where a match to an approved software product is <u>not</u> located, the SAM Tool returns that result. This information may support a security policy decision, such as whether to permit a network connection from the endpoint (e.g., missing a required antivirus product.)
6. Where a match to a prohibited software product <u>is</u> located, the SAM Tool returns that result. This information may support another type of security policy decision, such as quarantining a device that is found to contain a prohibited software product.

5.2.5.3 Outcomes

For the process described above, the application of SWID tags enables the organization to use automation for the accurate and timely reporting of software inventory information. While this capability exists without SWID tags, the consistent and precise information these tags provide is beneficial.

5.2.6 Usage Scenario 8: 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 is not the case. SWID tags provide a comprehensive, compact description of installed software, 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., NVD, vendor alerts, US-CERT alerts) and the SAM tools that collect inventory information.

If a software provider uses additional information to identify the software product (e.g., Professional Edition), this additional data will be included in the bulletin to match SWID tag data, using the `<Meta>` element providing at least the `@product`, `@productFamily`, and `@revision` attributes.

5.2.6.1 Assumptions

This usage scenario assumes the existence of an enterprise repository, populated with collected SWID tags that are created in accordance with the requirements described in Section 4.

5.2.6.2 Process

1. Using the information about reported software vulnerabilities from one or more software vulnerability bulletins, the SAM tool reviews each SWID tag record.

2. Where a record exists that matches the query parameters, 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 the vulnerability appears to have been patched. This information can be reported for security situational awareness and it also supports security analysis.

5.2.6.3 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.

5.2.7 Usage Scenario 9: Detection of Media/Software Tampering

An important element of software asset management is the discovery of software tampering, either pre-installation or post-installation.

In the first instance, the contents of a corpus tag (e.g., digital signature, file/size/hash values in the `<Payload>` element) may be compared to the actual media contents. In the second instance, the known tag values of installed software/packages (from the local endpoint repository and/or the enterprise repository) may be compared to the files observed on the endpoint.

Detection of potential tampering may be used for several purposes:

- To prevent installation from suspect media;
- To report, as part of a SAM report, potential tampering of an endpoint;
- To quarantine an endpoint pending further investigation; or,
- To 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.

This usage scenario provides an example of the benefit of a local repository that works in concert with an enterprise repository. The local endpoint is able to perform a comparison of the recorded cryptographic hash to the observed local file quickly enough to enable such a check on demand. Because some legacy cryptographic hash algorithms are easily spoofed, the use of a stronger methodology as described in Section 3.5 will help provide confidence in the findings. Comparison of observed hash values with recorded values in the enterprise repository requires additional network/computing resources and is more commonly performed as a periodic monitoring task.

5.2.7.1 Assumptions

This usage scenario assumes the existence of an enterprise repository, populated with collected SWID tags that are created in accordance with the requirements described in Sections 3 and 4.

5.2.7.2 Process

1. For each endpoint, the SAM Tool reads the stored cryptographic hashes for each file listed in `<Payload>` or `<Evidence>` elements, `@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 SAM Tool 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.

5.2.7.3 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 other malicious activities.

5.3 Actions Based on SWID Tag Query Results

Based upon the results of the query scenarios described above, the following is an example of a type of activity recommended to achieve security objectives.

5.3.1 Usage Scenario 10: 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.

5.4 Additional Usage Scenarios

Many of the scenarios described above are useful for information systems managers who are using SWID tags in a production environment. The ability to consistently and accurately reference software inventory information helps vendors and other SWID users to improve automation and interoperability for their customers and constituents.

Appendix A—Forming CPE Names

This appendix presents an algorithm for forming CPE names using attributes contained within a software product's primary tag.

A CPE name can be mechanically generated according to the following rules in Augmented BNF syntax [RFC 5234]:

```
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:*:*:*:*:*
```

Appendix B—Acronyms

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

ABNF	Augmented Backus–Naur Form
API	Application Programming Interface
CA	Certificate Authority
CIO	Chief Information Officer
CISO	Chief Information Security Officer
COTS	Commercial-Off-the-Shelf
CPE	Common Platform Enumeration
CVE	Common Vulnerabilities and Exposures
DSS	Digital Signature Standard
FIPS	Federal Information Processing Standards
HTML	Hypertext Markup Language
ID	Identifier
IEC	International Electrotechnical Commission
IETF	Internet Engineering Task Force
IP	Internet Protocol
ISBN	International Standard Book Number
ISCM	Information Security Continuous Monitoring
ISO	International Organization for Standardization
IT	Information Technology
ITL	Information Technology Laboratory
NAS	Network-Attached Storage
NIST	National Institute of Standards and Technology
NISTIR	National Institute of Standards and Technology Internal Report
NVD	National Vulnerability Database
RFC	Request for Comment
RPM	RPM Package Manager
SAM	Software Asset Management
SCAP	Security Content Automation Protocol
SD	Secure Digital
SHA	Secure Hash Algorithm
SHS	Secure Hash Standard
SIEM	Security Information and Event Management
SP	Special Publication
SWID	Software Identification
URI	Uniform Resource Identifier
US	United States
USB	Universal Serial Bus
US-CERT	United States Computer Emergency Readiness Team
USG	United States Government
W3C	World Wide Web Consortium
XAdES	XML Advanced Electronic Signature
XAdES	XML Advanced Electronic Signature with Time-Stamp
XML	Extensible Markup Language

XPath	XML Path Language
XSD	XML Schema Definition

Appendix C—References

[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].

[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].

[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].

[ISO/IEC 19770-5:2013] International Organization for Standardization/International Electrotechnical Commission, *Information technology -- Software asset management -- Part 5: Overview and vocabulary*, ISO/IEC 19770-5:2013, 2013. https://www.iso.org/obp/ui/#iso:std:iso-iec:19770:-5:ed-1:v1:en [accessed 6/10/15].

[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 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].

[RFC 5234] Crocker, D., and Overell, P. (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].

[SEMVER] Preston-Werner, T. *Semantic Versioning Specification 2.0.0.* http://semver.org/spec/v2.0.0.html [accessed 7/2/2015].

[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].

[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].

[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].

[XPATH 2.0] World Wide Web Consortium (W3C) (2010) XML path language (XPath) 2.0. http://www.w3.org/TR/xpath20 [accessed 6/10/15]

Appendix D—Change Log

Release 1 – May 29, 2015 (Initial public comment draft)

Release 2 – July 22, 2015 (Second public comment draft)

Functional Additions/Changes/Removals:

- Greatly expanded the Section 2.2 introduction to incorporate the software lifecycle and explain its support by SWID tags.
- Added material to Sections 2.2.2 and 2.2.3 to discuss the `<Link>` element and its role in documenting relationships between products and between patches, respectively.
- Added Table 1 to Section 2.4.1 to better explain how tag types may be determined.
- Expanded Section 2.4.4 to enumerate values that the `<Link>` element `@href` attribute can point to.
- Created a new Section 3.3 on implementing `<SoftwareIdentity>` elements, and created a new GEN-2 guidance item. Renumbered all the other guidance items in the rest of Section 3.
- Expanded GEN-3 (formerly GEN-2) in Section 3.4 (formerly Section 3.3) to add a recommendation for second-party authoritative tag creators.
- Split the (formerly) GEN-5 guidance item on file hash values into two items: GEN-6 (for authoritative tag creators) and GEN-7 (for non-authoritative tag creators).
- Split the (formerly) GEN-6 guidance item on SHA-256 file hashes into two items: GEN-9 (for authoritative tag creators) and GEN-10 (for non-authoritative tag creators).
- Added GEN-12 guidance item on SHA-512 hash function performance on 64-bit systems.
- Expanded the discussion in Section 3.6 (formerly Section 3.5) on implementing digital signature; this included adding a new guidance item, GEN-13.
- Created a new Section 3.7 on using tag identifiers to refer to product installation packages, product releases, and product patches. Included adding new guidance items, GEN-14, GEN-15, GEN-16, and GEN-17.
- Rewrote Section 3.8 (formerly Section 3.6) on updating tags. Deleted GEN-9 and GEN-10 guidance items, added GEN-18 and GEN-19.
- Added Section 4.1.1 on specifying the version and version scheme in corpus tags. Included adding new guidance items, COR-1, COR-2, and COR-3.
- Added Section 4.2.1 on specifying the version and version scheme in primary tags. Included adding new guidance items, PRI-1 through PRI-5.
- Softened PRI-7 (formerly PRI-2) to use SHOULD language instead of MUST.
- Moved the rules for mechanically generating CPE names from Section 4.2.3 (formerly Section 4.1.2) to Appendix A.
- Rewrote Section 4.4.1 (formerly Section 4.2.2), including major changes to SUP-1 (formerly SUP-2).
- Greatly expanded Section 4.4.2 (formerly Section 4.2.1), and rewrote SUP-2 (formerly SUP-1).

- Rewrote the Section 5 introduction.
- Rewrote Section 5.1 and both of its usage scenarios.
- Added a new Section 5.2 (based largely on the former Section 5.1.2) on using SWID tags, along with the following usage scenarios:
    - New Section 5.2.1, identifying instances of an installed product or patch
    - New Section 5.2.2, identifying endpoints that are missing a product or patch
    - New Section 5.2.3, identifying orphaned software, shared components, and patches on endpoints
    - New Section 5.2.4, preventing installation of prohibited software
    - New Section 5.2.5, detecting installed instances of prohibited software
    - Section 5.2.6 (formerly Section 5.2), determining vulnerable software on an endpoint
- Rewrote Section 5.2.7 (formerly Section 5.3) on detection of media/software tampering
- Deleted Section 5.4 on mapping a SWID tag to other SWID schemes

Editorial Changes:

- Made minor editorial revisions throughout the report.
- Added a references appendix (Appendix C) and a change log appendix (Appendix D).
- Expanded the acronym list in Appendix B (formerly Appendix A).
- Rewrote and reorganized Section 2.1 to be more clearly focused on tag placement.
- Reordered the Section 2.2 subsections to be in a more logical order:
    - Corpus moved from 2.2.4 to 2.2.1
    - Primary moved from 2.2.1 to 2.2.2
    - Patch stayed at 2.2.3
    - Supplemental moved from 2.2.2 to 2.2.4.
- Added a summary of the guidance category abbreviations to the Section 3 introduction.
- Reordered the material within Section 3.4.
- Reordered the Section 4 subsections to be in a more logical order:
    - Corpus moved from 4.4 to 4.1
    - Primary moved from 4.1 to 4.2
    - Patch stayed at 4.3
    - Supplemental moved from 4.2 to 4.4.
- Switched the order of the Section 4.4 (formerly Section 4.2) subsections on implementing supplemental tags.
- Switched the order of the Section 4.5 summary key points to correspond to the new sequence of Section 4.