The attached DRAFT document (provided here for historical purposes) has been superseded by the following publication:

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

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

Publication Date: **April 2016**

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

NISTIR 8060 (Third DRAFT)

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

David Waltermire
Brant A. Cheikes
Larry Feldman
Greg Witte

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

NISTIR 8060 (Third DRAFT)

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

David Waltermire
Computer Security Division
Information Technology Laboratory

Brant A. Cheikes
The MITRE Corporation
Bedford, Massachusetts

Larry Feldman
Greg Witte
G2, Inc.
Annapolis Junction, MD

August 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
89 pages (August 2015)

Certain commercial entities, equipment, or materials may be identified in this document in order to describe an experimental procedure or concept adequately. Such identification is not intended to imply recommendation or endorsement by NIST, nor is it intended to imply that the entities, materials, or equipment are necessarily the best available for the purpose.

There may be references in this publication to other publications currently under development by NIST in accordance with its assigned statutory responsibilities. The information in this publication, including concepts and methodologies, may be used by Federal agencies even before the completion of such companion publications. Thus, until each publication is completed, current requirements, guidelines, and procedures, where they exist, remain operative. For planning and transition purposes, Federal agencies may wish to closely follow the development of these new publications by NIST.

Organizations are encouraged to review all draft publications during public comment periods and provide feedback to NIST. All NIST Computer Security Division publications, other than the ones noted above, are available at http://csrc.nist.gov/publications.

Public comment period: *August 31, 2015* through *September 24, 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 third discussion draft of this report. The authors are conducting a number of iterations of this report 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 four to six iterations of this cycle before finalizing this report. 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 cover the overall report, noting three areas of particular interest:

- The clarity and feasibility of the guidelines in Sections 3 and 4
- Section 5, which has been reorganized and largely rewritten
- Appendix A, which has been completely rewritten

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

Any mention of commercial products or reference to commercial organizations is for information only; it does not imply recommendation or endorsement by NIST, nor does it imply that the products mentioned are necessarily the best available for the purpose.

All names are trademarks or registered trademarks of their respective owners.

Document Conventions

This report 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 report 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 report 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 report 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 report 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 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 frequently 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, and associated files in device storage areas. 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 tag implementation guidelines that supplement the SWID tag specification. Lastly, it presents a set of operational usage scenarios, which illustrate how SWID tags conforming to these guidelines can be used to achieve a variety of cybersecurity goals. By following the guidelines in this report, tag creators can have confidence they are providing all the necessary data, with the requisite data quality, 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 report 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 because 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 report 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 report 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 report 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.
- SWID 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 report 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 are specific 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 acronyms used in this report.
- Appendix C provides the references for the report.
- Appendix D provides 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 report 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 2.3.2 of this report). This restriction is necessary to ensure that any supplied digital signatures and thumbprints used to authenticate SWID tags remain valid and usable (see Section 2.4). 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 2.1.4).

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 describes the four types of SWID tags and the distinct roles they play at key points in the software lifecycle. Section 2.2 discusses expectations regarding where SWID tags reside relative to the products they identify, how the location of a tag may or may not relate to the computing device(s) where the tagged product may be executed, and how tags are deployed to devices. Section 2.3 presents an overview of the basic data elements that comprise a SWID tag. Section 2.4 discusses how SWID

[1] On devices that have filesystems, 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 filesystems, tags should be accessible through platform-specific interfaces and/or maintained in platform-specific storage locations.

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

2.1 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. Figure 1 illustrates the steps in the software lifecycle, and the relationship between those lifecycle events and the four types of SWID tags.

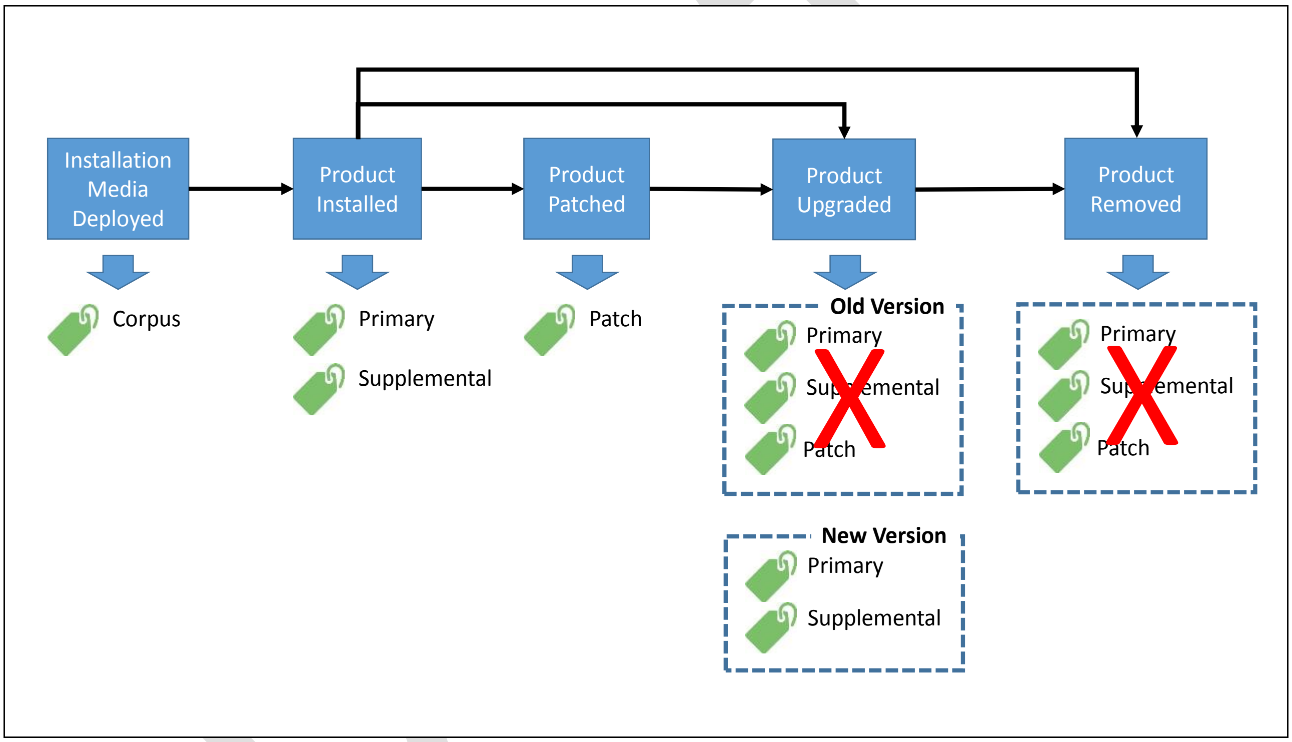

Figure 1: SWID Tags and the Software Lifecycle

These software lifecycle events and their associated tags are discussed in the following subsections.

2.1.1 Corpus Tags

Before software is installed, it is typically delivered or otherwise made available to an endpoint in the form of a software 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. 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 2.3.6 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.

2.1.2 Primary Tags

As illustrated in Figure 1, primary tags are involved in different software lifecycle events. The SWID specification defines *primary tags* for vendors and distributors to use to identify and describe software products once they have been successfully installed on an endpoint. The primary tag for each tagged product furnishes, at a minimum, values for all data elements that are designated “mandatory” in the SWID specification. 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. When a product is upgraded, the primary tag associated with the old version is removed and replaced with a primary tag for the new version. When a product is removed from a device, its primary tag is removed as well.

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 2.3.4), which may be used within a SWID tag to document relationships between the product described by the tag and other products or items that may be available. Three types of relationships are worth noting here:

- **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 has a separately installable software product as one of its components, the product's primary tag points to the primary tag of the component product 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`.

The relationships that may be expressed in primary tags are illustrated in Figure 2.

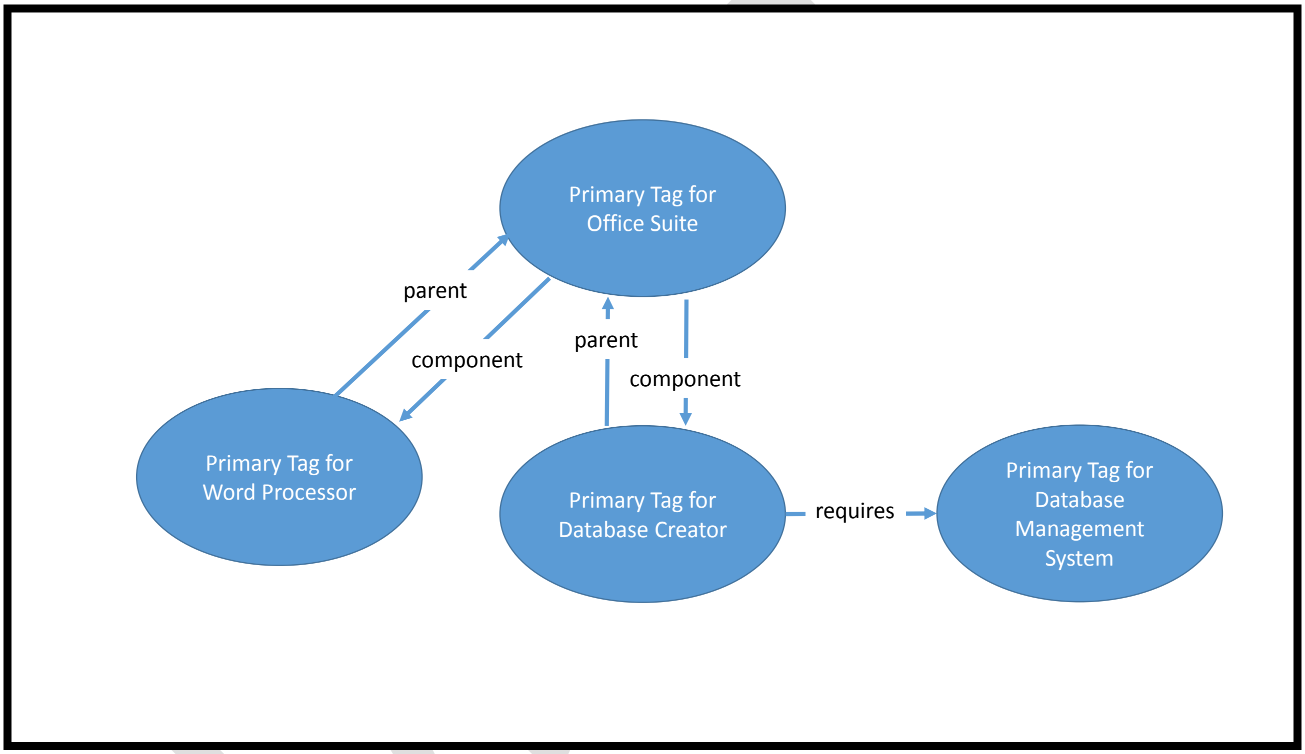

Figure 2: Primary Tag Relations

2.1.3 Patch Tags

From time to time after a software product is installed, the software provider may release patches to correct errors or add new features. Patches make localized changes to the installed product's codebase. Such localized changes may be tracked separately from the base product. The SWID specification defines *patch tags* for vendors and distributors to use to identify and describe each 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.)

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 2.3.4 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 2.3.6 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.

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 that may be expressed in patch tags are illustrated in Figure 3. 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.

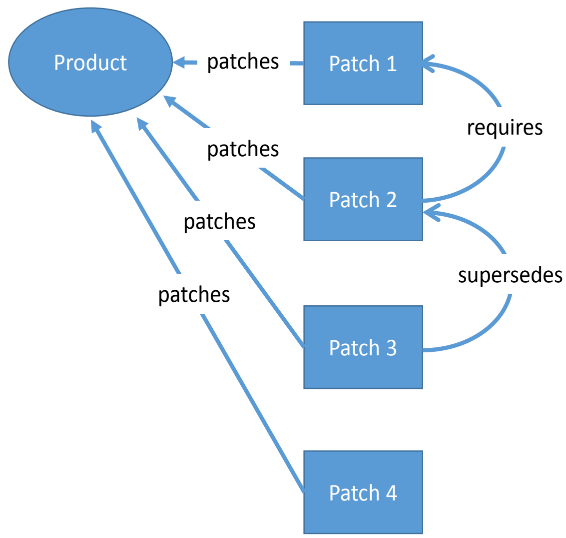

Figure 3: Patch Tag Relations

2.1.4 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 and contact information for local responsible parties.

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.

The relationships that may be expressed in supplemental tags are illustrated in Figure 4.

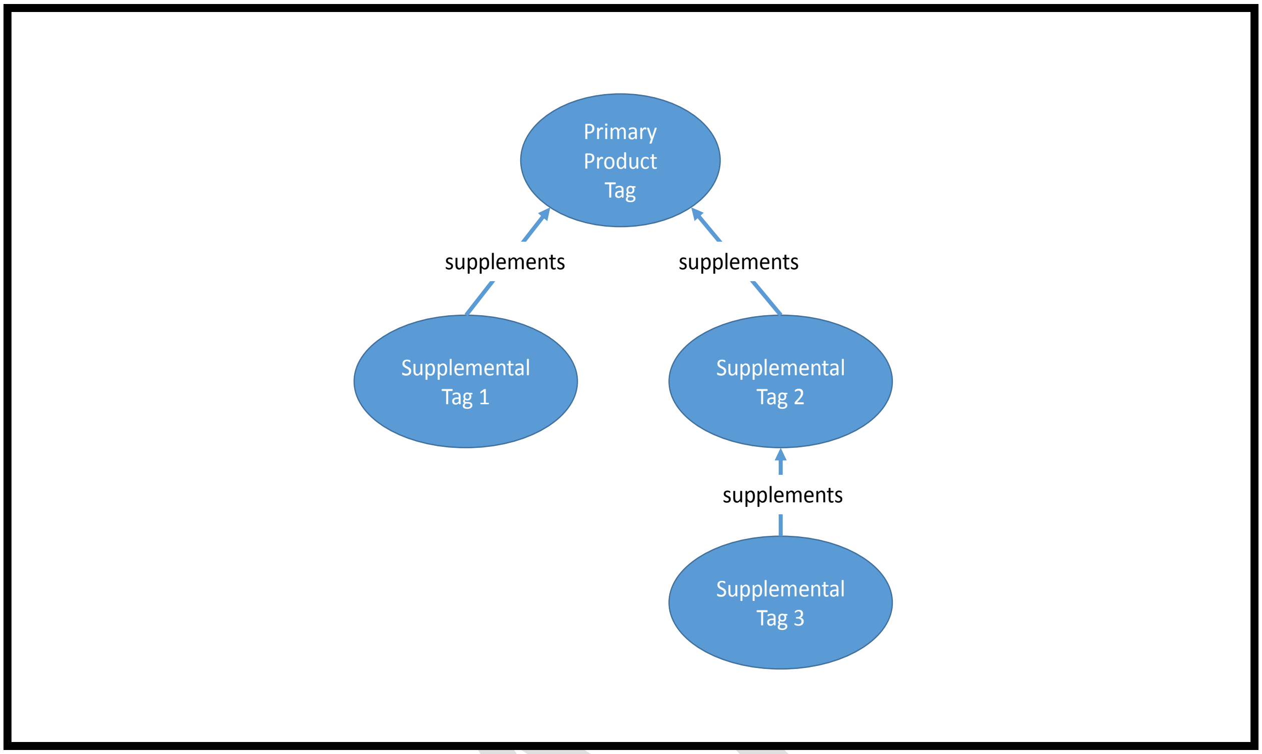

Figure 4: Supplemental Tag Relations

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 of the product's other 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.

As Figure 1 illustrates, after a software product is upgraded, all primary, patch, and supplemental tags associated with the pre-upgrade version of the product should be removed. If needed, new supplemental tags associated with the upgraded version may be deployed. When a software product is removed, all primary, patch, and supplemental tags associated with the product should be removed.

2.2 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 st*ored* 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 report 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 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.

---

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

The 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.

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 do the following:

- Confirm that a tag that has been discovered on an endpoint has not been modified
- Restore a tag that has been inadvertently deleted
- Correct a tag that has been improperly modified
- 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 needed 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.

2.3 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 2.1 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 2.3.1. The following sub-elements are used to express distinct categories of product information: `<Entity>` (Section 2.3.2), `<Evidence>` (Section 2.3.3), `<Link>` (Section 2.3.4), `<Meta>` (Section 2.3.5), and `<Payload>` (Section 2.3.6).

2.3.1 <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 report, guidelines are provided that **require** 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 Sections 4.1.1 and 4.2.1 of

this report provide guidelines that **require** a value for `@versionScheme` in corpus and primary tags, respectively.)

- `@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 changed to indicate that errors in an earlier tag have been corrected, or that new information has been added. A value for `@tagVersion` is **optional** and defaults to 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 and have the same `@tagId` but 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*. This report provides guidelines requiring that at most one of `@corpus`, `@patch`, or `@supplemental` be set to true (see Sections 4.1.1, 4.2.1, 4.3.1, and 4.4.1). In Sections 4.3.1 and 4.4.1 of this report, 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

2.3.1.1 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 2.3.2) 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>
```

2.3.1.2 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 2.3.2) is included so the example illustrates all data values required in a minimal tag that conforms to the standard. The <Link> element (Section 2.3.4) is included to illustrate how a supplemental tag may be associated with the primary tag shown above in Section 2.3.1.1. 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>
```

2.3.1.3 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>
```

2.3.2 <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), or distributor(s), 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. The `@role` attribute can list multiple roles with a space separating each role. 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>
```

2.3.3 <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 2.3.6). 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>
```

2.3.4 <SoftwareIdentity> Sub-Element: <Link>

Modeled on the HTML [LINK] element, SWID tag <Link> elements are used to record a variety of relationships between tags and other items. A typical use of the <Link> element is to document a relation that exists between a product or patch described by a *source tag* (the tag containing the <Link> element) and a product or patch described by a *target tag* (the tag to which the <Link> element points). <Link> elements may also be used to associate a source tag with other arbitrary information elements.

A <Link> element is often used to associate a patch tag or supplemental tag to a primary tag (see Sections 2.1.3 and 2.1.4). Other uses include pointing to documents containing applicable 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.

Table 2 lists the pre-defined values of the `@rel` attribute defined in the SWID specification. Note that this list may be extended to support future needs.

Table 2: <Link> Relations

Relation	Meaning
ancestor	Defines a link to an ancestor of the product. This relation may be used to indicate a pre-upgrade version of the product.
component	Defines a link to a component of the product. A component could be an independently functioning application that is part of a product suite or bundle, as well as a shared library, language pack, etc.
feature	Defines a link to a part of the product that can be enabled or disabled separately without necessarily modifying any physical files.
installationmedia	Defines a link to the installation media used to install the software product.
packageinstaller	Defines a link to a tool or entity required to install the product.
parent	Defines a link to the parent of the product.
patches	Defines a link to the product to which the patch was applied.
requires	Defines a link to a required patch, or to any other software product that is required in order for the product described by the source tag to function properly.
see-also	Defines a link to other software products that may relate in some manner to the software identified in the source tag. Such other products might be add-ons or extensions that may be of interest to the user/administrator of the device.
supersedes	Defines a link to a superseded patch.
supplemental	Defines a link to a supplemental tag.
<any>	Additional relationships can be specified by referencing the Internet Assigned Numbers Authority (IANA) Link Relations registration library.[3]

In addition to `@href` and `@rel`, the SWID specification defines several other optional attributes of `<Link>` elements to support specialized needs. The usage scenarios and requirements relating to these optional attributes have not yet been developed, so they will not be discussed further here.

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"
```

[3] See http://www.iana.org/assignments/link-relations/link-relations.xhtml for the current list of defined link relations.

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

The patch in this example is linked to the patched product's tag using that product's `@tagId`.

2.3.5 <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>
```

2.3.6 <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>` lists 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 are not installed on the device.)

The `<Payload>` element is a container for `<Directory>`, `<File>`, `<Process>`, and/or `<Resource>` elements, similar to the `<Evidence>` element (Section 2.3.3). When the `<Payload>` element is used, the information contained in the element is considered to be authoritative information about the software. This differs from the use of the `<Evidence>` element, which is used to store results from a scan that indicate why the product is believed to be installed.

The following example illustrates a primary tag with a `<Payload>` element 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>
```

2.4 Authenticating SWID Tags

Because SWID tags are XML 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.[4] Users may also include a hexadecimal hash string (the "thumbprint") to document the relationship between the tag entity and the signature, using the `<Entity> @thumbprint` attribute.

Section 6.1.10 of the 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

---

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

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].

Detailed guidelines pertaining to the implementation and use of digital signature standards to sign and validate SWID tags remain under review and will be presented in a future release of this report.

2.5 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>
```

2.6 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 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 guideline in the next two sections is prefixed with a coded identifier for ease of reference. Such identifiers have the following format: *CAT-NUM*, where "CAT" is a three-letter symbol indicating the guidance category, and NUM is a number. Guidelines are grouped into the following categories:

- **GEN:** General guidelines applicable to all types of SWID tags
- **COR:** Guidelines specific to corpus tags (see Section 4.1)
- **PRI:** Guidelines specific to primary tags (see Section 4.2)
- **PAT:** Guidelines specific to patch tags (see Section 4.3)
- **SUP:** Guidelines specific to supplemental tags (see Section 4.4)

This section provides implementation guidelines that address issues common to all situations in which tags are deployed and processed. Section 4 provides guidelines that vary according to the type of tag being implemented.

3.1 Limits on Scope of Guidelines

This report 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.

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

All guidelines in this report are intended solely to extend and not to conflict with any guidelines provided by the SWID specification. Guidelines in this report either:

- Strengthen existing guidelines contained in the SWID specification by elevating "SHOULD" clauses contained in the SWID specification to "MUST" clauses, or
- Add guidelines to address implementation issues where the SWID specification is silent or ambiguous by adding new "SHOULD" or "MUST" clauses.

In no cases should this report's guidelines be construed as either weakening or eliminating existing guidelines 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) 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 that tag.

Tags may be created by authoritative or non-authoritative entities. For the purposes of this report, 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 using reverse engineering methods. As a shorthand, this report uses the term "authoritative tag" to refer to tags created by authoritative entities, and "non-authoritative tag" to refer to tags created by non-authoritative entities.

Unless otherwise specified, guidelines in this report are directed at both authoritative and non-authoritative tag creators. Guidelines prefixed with "[Auth]" are directed specifically at authoritative tag creators, and guidelines prefixed with "[Non-Auth]" are directed specifically at non-authoritative tag creators.

3.3 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 one @role of "tagCreator", tag consumers must 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 a single <Entity> element that indicates that the entity in the tagCreator role is also in one of any of these additional predefined roles: "aggregator", "distributor", "licensor", or "softwareCreator".

GEN-2. [Auth] Authoritative tag creators MUST provide an `<Entity>` element where the `@role` attribute contains the value `tagCreator` and at least one of these additional role values: "`aggregator`", "`distributor`", "`licensor`", or "`softwareCreator`".

If this guideline is observed, tag consumers may reliably distinguish between authoritative and non-authoritative tags according to this rule: If a tag contains an `<Entity>` element with a `@role` value that includes "`tagCreator`" as well as any of "`aggregator`", "`distributor`", "`licensor`", or "`softwareCreator`", then the tag is authoritative, otherwise it is non-authoritative.

The SWID specification only requires values for `@name` and `@regid` attributes in the `<Entity>` element where the `@role` attribute contains the value "`tagCreator`". Although the tag creator is not necessarily the same as the software creator, authoritative tag creators are expected to know the name and regid of the software creator. Because explicit knowledge of the software creator is important for many software inventory scenarios, authoritative tag creators are expected to furnish information on the software creator's identity.

GEN-3. [Auth] Authoritative tag creators MUST provide an `<Entity>` element where the value of the `@role` contains "`softwareCreator`", and the `@name` and `@regid` attribute values are also provided.

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.

3.4 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 by 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.

Similarly, knowing a file's version as recorded in the filesystem can be useful when searching for installed products containing a file with a known version. This motivates the following guideline:

> **GEN-6.** Every `<File>` element provided within a `<Payload>` or `<Evidence>` element MUST include a value for the `@version` attribute, if one exists for the file.

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-7.** [Auth] Every `<File>` element within a `<Payload>` element MUST include a hash value.

> **GEN-8.** [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-9.** 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 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 extremely 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 guidelines:

> **GEN-10.** [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-11.** [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-12.** Whenever a `<Payload>` or `<Evidence>` element 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, 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 be a better-performing integrity assurance measure.

3.5 Implementing Digital Signatures

This section contains 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-13.** 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.

3.6 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 2.1.1, 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 guideline.

> **GEN-14.** 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-15.** 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-16.** 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-17. 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.

3.7 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 3.6, 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 guideline.

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>` `@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-19. 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 guideline 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.

3.8 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.

- Question: 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?

3.9 Summary

These are the key points from this section:

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

- The intent of this guidance is to be broadly applicable to common IT usage scenarios that are relevant to all software providers and consumers.

- This section provided implementation guidance that addresses issues common to all situations in which tags are deployed and processed. The next section provides guidelines that vary according to the type of tag being implemented.

4 Implementation Guidance Specific to Tag Type

This section provides implementation guidelines that are specific to each of the four tag types defined in Section 2.1: 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 in Section 2.1.1, corpus tags are used to identify and describe products in a pre-installation state. This section provides guidance addressing the following topics related to implementation of corpus tags: setting the `<SoftwareIdentity>` `@corpus` attribute (Section 4.1.1), specifying `@version` and `@versionScheme` (Section 4.1.2), specifying `<Payload>` element information (Section 4.1.3), and signing corpus tags (Section 4.1.4).

4.1.1 Setting the <SoftwareIdentity> @corpus Attribute

To indicate that a tag is a corpus tag, tag implementers set the value of the `<SoftwareIdentity>` `@corpus` attribute to true. The SWID specification does not specifically prohibit tag implementers from also setting other tag type indicator attributes to true (e.g., `<SoftwareIdentity>` `@patch` and `<SoftwareIdentity>` `@supplemental`), but doing so would create confusion regarding how the information contained within the tag should be interpreted. This report provides guidelines to ensure that at most one tag type indicator attribute is set to true.

> **COR-1.** If the value of the `<SoftwareIdentity>` `@corpus` attribute is set to true, then the values of `<SoftwareIdentity>` `@patch` and `@supplemental` MUST be set to false.

4.1.2 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 report 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-2.** 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 report to fully resolve those matters. Instead, the following guidelines are 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-3.** If a corpus tag contains a value for the `<SoftwareIdentity> @version` attribute, it MUST also contain a value for the `<SoftwareIdentity> @versionScheme` attribute.

> **COR-4.** Whenever a value for the `<SoftwareIdentity> @versionScheme` attribute is provided in a corpus tag, it SHOULD be selected from a well-known public list of version scheme identifiers.

> **COR-5.** Publicly-listed version schemes intended for reference from within corpus tags SHOULD specify semantics for each version scheme sufficient for comparing two versions and determining their relative order in a sequence.

4.1.3 Specifying the 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 sizes and hash values.

> **COR-6.** A corpus tag MUST contain a `<Payload>` element that MUST enumerate every file that is included in the tagged installation media.

4.1.4 Signing a Corpus Tag

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.

- Question: 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: setting tag type indicator attributes to designate a tag as a primary tag (Section 4.2.1), specifying version and version scheme information (Section 4.2.2), specifying `<Payload>` or `<Evidence>` information (Section 4.2.3), and specifying attributes needed to form CPE names (Section 4.2.4).

4.2.1 Setting the <SoftwareIdentity> Tag Type Indicator Attributes

To indicate that a tag is a primary tag, tag implementers ensure that the values of all three tag type indicators (the `<SoftwareIdentity>` `@corpus`, `@patch`, and `@supplemental` attributes) are set to false. This is enforced by the following guideline.

PRI-1. To indicate that a tag is a primary tag, the `<SoftwareIdentity>` `@corpus`, `@patch`, and `@supplemental` attributes MUST be set to false.

4.2.2 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, guidelines for primary tags must distinguish between authoritative and non-authoritative primary tag creators.

PRI-2. [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-3. [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-4. [Non-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 if it can be determined.

PRI-5. [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-6. Whenever a value for the <SoftwareIdentity> @versionScheme attribute is provided in a primary tag, it SHOULD be selected from a well-known public list of version scheme identifiers.

PRI-7. Publicly-listed version schemes intended for reference from within primary tags SHOULD specify semantics for each version scheme sufficient for comparing two versions and determining their relative order in a sequence.

4.2.3 Specifying 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-8. [Auth] A <Payload> element MUST be provided, in either a software product's primary tag or a supplemental tag.

PRI-9. [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-10. <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 considered machine instruction files.

PRI-11. [Auth] The <Payload> element MUST list every machine instruction file comprising the product described by the tag.

PRI-12. [Non-Auth] The <Evidence> element MUST list every machine instruction file comprising the product described by the tag.

4.2.4 Specifying Product Metadata Needed for Targeted Search

The SWID specification furnishes the `<SoftwareIdentity> @name` attribute to capture "the software component name as it would typically be referenced." This is also called the product's *market name*, i.e., the product name as used on websites and in advertising materials to support marketing, sales, and distribution. Market names for commercial software products often combine a variety of market-relevant descriptive elements, including:

- **The product's "base name" distinguished from the provider's "brand name."** When, for example, the software provider whose legal name is "Acme Systems Incorporated" markets its "Roadrunner" product, it might use "Acme" as a company brand name prefixed to the base name of its products, as in "Acme Roadrunner."

- **The product's "market version."** On occasion, software providers distinguish between the version they assign to a product's underlying codebase (e.g., 5.6.2) and the version they assign to it for marketing purposes (e.g., 2015). For example, Acme Systems Incorporated might release codebase version 6.0 of their Roadrunner product with the market version of 2015. The market name for this product might then appear as "Acme Roadrunner 2015".

- **The product's "edition."** Some software providers market the same core product to different user audiences, selectively adding and/or removing features depending on their appeal to each audience. When this is done, providers may add an "edition" descriptor to the product's market name. For example, Acme might market a full-featured "Roadrunner" product to large companies, and refer to that product as the "Enterprise Edition." A stripped-down and less-costly instantiation of that product might be tailored to individual use on home computers, and designated the "Home Edition." As a result, two different market names might be used: "Acme Roadrunner 2015 for Enterprises" and "Acme Roadrunner 2015 for Home Offices."

- **The product's "revision."** In some specialized cases, for example, when a particular product receives unwanted attention due to defects, a software provider may be motivated to revise a product's market name in conjunction with the issuance of a major patch or product upgrade. When this happens, the revised market name might incorporate phrases such as "Service Release x" or "Revision y"; e.g., "Acme Roadrunner 2015 for Enterprises Service Release 2".

While any or all of these elements may be present in a product's market name and thus should appear in the `<SoftwareIdentity> @name` attribute of the product's primary tag, there is no consistency in whether or how those elements are included, making it difficult for a machine to reliably parse them out of the market name.

The problem is that these metadata elements—the product's base name exclusive of any provider name prefixes, its market version distinct from any codebase version, its edition, and its revision—are often needed by local administrators, cybersecurity personnel, and supporting automated tools when performing targeted searches. For example, a security advisory might announce that a major vulnerability has been discovered in the "Enterprise" edition of a product, while the “Home” edition is unaffected. As another example, an organization might want to declare and enforce a policy that only the “Enterprise” edition of Acme’s “Roadrunner” project may be installed on network devices, and that allowed installations are further restricted to the “Service Pack 2” revision of the product. To make this possible, there needs to be a way to individually refer to each descriptive element embedded within a product’s market name.

To accommodate this need, the SWID specification defines the following `<Meta>` element attributes:

- `@product`: This attribute provides the base name of the product. The base name is expected to exclude substrings containing the software provider’s name, as well as any indicators of the product’s version, edition, or revision level.

- `@colloquialVersion`: This attribute provides the market version of the product. This version may remain the same through multiple releases of a software product, whereas the version specified in the `<SoftwareIdentity>` `@version` is more specific to the underlying software codebase and will change for each software release.

- `@edition`: This attribute provides the edition of the product.

- `@revision`: This attribute provides an informal designation for the revision of the product.

If these attributes are specified, not only will targeted searches be easier to define and execute, but also it will be possible to mechanically generate a valid CPE name from an input SWID tag. (See Appendix A for an algorithm that may be used to generate such CPE names.)

The guideline is as follows:

> **PRI-13.** If appropriate values exist and can be determined, a `<Meta>` element MUST be provided and MUST furnish values for as many of the following attributes as possible: `@product`, `@colloquialVersion`, `@revision`, and `@edition`.

4.3 Implementing Patch Tags

As noted earlier in Section 2.1.3, a patch tag is used to describe localized changes applied to an installed product’s codebase. This section provides guidance addressing the following topics related to implementation of patch tags: setting the `<SoftwareIdentity>` `@patch` attribute (Section 4.3.1), linking patch tags to related tags (Section 4.3.2), and specifying `<Payload>` or `<Evidence>` information (Section 4.3.3).

4.3.1 Setting the <SoftwareIdentity> @patch Attribute

To indicate that a tag is a patch tag, tag implementers set the value of the `<SoftwareIdentity>` `@patch` attribute to true. The SWID specification does not specifically prohibit tag implementers from also setting other tag type indicator attributes to true (e.g., `<SoftwareIdentity>` `@corpus` and `<SoftwareIdentity>` `@supplemental`), but doing so would create confusion regarding how the information contained within the tag should be interpreted. This report provides guidelines to ensure that at most one tag type indicator attribute is set to true.

PAT-1. If the value of the `<SoftwareIdentity>` `@patch` attribute is set to true, then the values of `<SoftwareIdentity>` `@corpus` and `<SoftwareIdentity>` `@supplemental` MUST be set to false.

4.3.2 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 guidelines are 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-2. 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 guideline 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-3. [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 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 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-4. [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.

4.3.3 Specifying 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.

Guidelines in this section propose that patch tags document three distinct types of changes:

1. **Modify:** 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 modified or added, patch tags must include file sizes 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-5. [Auth] A patch tag MUST contain a `<Payload>` element that MUST enumerate every file that is modified, removed, or added by the patch.

PAT-6. [Auth] Each `<File>` element contained within the `<Payload>` element of a patch tag MUST include an extension attribute named `@patchEvent`, which MUST have one of the following values:

- The string value "`modify`" 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-7. [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

As noted in Section 2.1.4, supplemental tags are used for any purpose to furnish identifying and descriptive information not contained in other tags. This section provides guidance addressing the following topics related to implementation of supplemental tags: setting the `<SoftwareIdentity> @supplemental` attribute (Section 4.4.1), linking supplemental tags to other tags (Section 4.4.2), and establishing the precedence of information contained in a supplemental tag (Section 4.4.3).

4.4.1 Setting the <SoftwareIdentity> @supplemental Attribute

To indicate that a tag is a supplemental tag, tag implementers set the value of the `<SoftwareIdentity> @supplemental` attribute to true. The SWID specification does not specifically prohibit tag implementers from also setting other tag type indicator attributes to true (e.g., `<SoftwareIdentity> @corpus` and `<SoftwareIdentity> @patch`), but doing so would create confusion regarding how the information contained within the tag should be interpreted. This report provides guidelines to ensure that at most one tag type indicator attribute is set to true.

SUP-1. If the value of the `<SoftwareIdentity> @supplemental` attribute is set to true, then the values of `<SoftwareIdentity> @corpus` and `<SoftwareIdentity> @patch` MUST be set to false.

4.4.2 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 other supplemental tags. Because the SWID specification does not clearly state how a supplemental tag should indicate its linkage to other tags, a clarifying guideline is provided here.

> **SUP-2.** 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.

4.4.3 Establishing 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 as `Foo`, and a supplemental tag is also present that is linked to the primary tag but specifies the value of the `<SoftwareIdentity>` `@name` attribute as `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 report 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 guideline.

SUP-3. 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 modified, 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 enable security professionals to manage a number of security and operational risks. This section describes how to use SWID tags to achieve three key cybersecurity objectives:

- Minimize exposure to publicly disclosed software vulnerabilities
- Enforce organizational policies regarding authorized software
- Control network resource access from potentially vulnerable endpoints

By using SWID tags in accordance with the guidelines provided in previous sections of this report, the security practitioner (e.g., Chief Information Security Officer (CISO), Information System Security Officer (ISSO)) is able to achieve these objectives quickly, accurately, and efficiently. The application of SWID tag capabilities provides consistent, and often automated, results to reduce cybersecurity risk. Table 3 at the end of this section illustrates the association among these guidelines and the usage scenarios.

The rest of this section is organized according to the three cybersecurity objectives described above:

- Section 5.1 describes how to minimize exposure to known vulnerabilities.
- Section 5.2 discusses the use of SWID tags for enforcing organizational software policies.
- Section 5.3 describes how to prevent potentially vulnerable endpoints from connecting to network resources.
- Section 5.4 associates these scenarios with the guidelines provided in earlier chapters.

Sections 5.1 through 5.3 each describe the cybersecurity objective to be achieved, followed by specific usage scenarios that contribute to achieving the objective. Each section is comprised of a description of its objective's relevance to the software lifecycle, the initial conditions that enable it, and the scenario process steps. The process steps also point to relevant guidelines from previous sections that enables the scenario, referenced parenthetically in **bold** within the process.

5.1 Minimizing Exposure to Publicly-Disclosed Software Vulnerabilities

This cybersecurity objective focuses on the use of SWID tags to help security practitioners minimize risks from exploitation of endpoints with known vulnerabilities within enterprise networks. To maintain an effective security posture, these practitioners need to maintain awareness of software products installed on enterprise networks, including software updates and patches. They need to maintain awareness of vulnerabilities related to this software, including those vulnerabilities for which a patch or other remediation is not yet available. Security practitioners also need to maintain awareness of changes to the software on each endpoint, since each change could intentionally or inadvertently introduce vulnerabilities to that endpoint. For example, a user might unintentionally roll back a patch that mitigated a critical vulnerability.

This section describes the following usage scenarios (USs):

- US 1 – Continuously Monitoring Software Inventory (Section 5.1.1)
- US 2 – Ensuring that Products are Properly Patched (Section 5.1.2)
- US 3 – Correlating Inventory Data with Vulnerability Data to Identify Vulnerable Endpoints (Section 5.1.3)
- US 4 – Discovering Vulnerabilities Due to Orphaned Software Components (Section 5.1.4)

5.1.1 US 1 – Continuously Monitoring Software Inventory

This scenario describes the use of SWID tags to gather and maintain an up-to-date and accurate accounting of software inventory on each endpoint, then aggregate that data, if needed, to regional and/or enterprise-wide repositories. Organizations are able to maintain an ongoing understanding of installed software inventory by continuously monitoring software change events. Information provided by SWID tags contributes to an up-to-date and accurate understanding of the software on endpoints. As software changes are made, the endpoint's software inventory is updated to reflect those changes. Modifications occur throughout the software lifecycle including installing, upgrading, patching, and removing software.

One or more software product discovery or monitoring tools (referred to in this section as a "discovery tool") can continuously **monitor** endpoints for software changes, either on an event-driven basis or through periodic assessment of installation locations. These tools **discover** changes, including modifications to existing SWID tags on the endpoint. This analysis should consider various sources for performing this discovery (see Section 2.2 for a discussion of SWID tag placement on devices), 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, Universal Serial Bus (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 which is executable from an endpoint.

SWID tags provide identification, metadata, and relationship information about an endpoint's installed software. For tagged software, SWID tags from authoritative sources can be used to identify installed software on the endpoint. 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 time of installation.

As the tools **collect** the data, SWID tags enable many reporting capabilities for enterprise system software inventories. SWID tags can be **aggregated** to one or more repositories (e.g., regional or enterprise) to enable accurate and reliable reporting of the software products installed on a set of organizational endpoints. This aggregation supports the exchange of normalized data pertaining to these products, an important component of effectively managing IT across an enterprise. SWID tags provide a vendor-neutral and platform-independent way to **analyze** the state of installed software (e.g., software installed, products missing, or software in need of patching) within the organization, and **monitor** endpoints to maintain continual awareness of the security posture.

5.1.1.1 Initial Conditions

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.
- SWID tags to be aggregated from local repositories have been created in accordance with the guidelines described in Sections 3 and 4.
- Some installed software products will not have an associated SWID tag because an authoritative source did not furnish a tag as part of the products' installation package.

5.1.1.2 Process

1. Upon detecting new or changed software in an installation location or a mounted filesystem on the endpoint, the discovery tool will attempt to discover appropriate SWID tags in that location. Changes to be detected may include:
   - New software items (or subcomponents) that were not previously in the inventory
   - Changes or updates to previously installed software products
   - New or modified SWID tags, as indicated by a new `@tagId` or `@tagVersion` attribute within the `<SoftwareIdentity>` element
2. The discovery tool will update the local endpoint repository with the data from the existing SWID tags, creating entries for software products and their components. Because the software version information is critical for understanding the configuration and potential vulnerabilities of the endpoint, if any primary tag contains such version information (using the `<SoftwareIdentity>` element's `@version` and `@versionScheme` attributes), then that information will be recorded (cf. **PRI-2, PRI-3**).

If any tags are identified as not being in compliance with the ISO 19770-2:2015 specification (cf. **GEN-1**), those tags will not be recorded in the database, since they may not be reliable for the purpose of software inventory.

3. The tool will determine the type of tag discovered, based upon the `<SoftwareIdentity>` `@corpus`, `@patch`, and `@supplemental` attributes (cf. **PRI-1, COR-1, PAT-1, SUP-1, GEN-17**). The discovery tool will read the payload information provided within the tag, including the files' names, sizes, and cryptographic hashes for each component of the software product. These values will later be used to perform file integrity verification (cf. **GEN-5, GEN-6, GEN-7, GEN-9, GEN-10, GEN-12, PRI-7, PRI-9)**.

4. The discovery tool will attempt to determine if the tag is authoritative by checking that the `@regid` of the `<Entity>` element containing the `@role` value "`tagCreator`" also contains the `@role` value of "`softwareCreator`", "`aggregator`", "`distributor`", or "`licensor`" (cf. **GEN-2**, **GEN-3**).

5. If a tag was not installed with the software, the discovery tool will create a non-authoritative tag on the endpoint for each instance of a discovered application.

   Information about the files discovered is important to support continuous monitoring for software vulnerabilities, so the created tag will list every machine instruction file comprising the software product discovered (see Section 4.2.3), using the `<Evidence>` element (cf. **PRI-9, PRI-10, PRI-12**). This information will include roles and files' names, sizes, versions, and cryptographic hashes discovered (cf. **GEN-4**, **GEN-5**, **GEN-6, GEN-8**, **GEN-9**, **GEN-11, GEN-12**). It will also include any version information determined for the software product (cf. **PRI-2**, **PRI-3**, **PRI-6, PRI-7)**.

   Where pre-2015 SWID tags are discovered, these tags are not stored in the repository, but their contents might be useful to support the evidence collected above.

6. Many cybersecurity decisions will be based upon the authenticity and integrity of the SWID tags discovered. To validate the integrity of the discovered tag, the discovery tool can confirm the digital signature as provided by the `@thumbprint` attribute of the `<Entity>` element (cf. **GEN-13**).

7. The discovery tool will read the tag identifier (i.e., `@tagId`) and the tag location, along with the type of tag discovered or created: corpus tags for pre-installation software (cf. **GEN-14**), primary tags for post-installation software (cf. **GEN-15)**, and patch tags for software patches (cf. **GEN-16**). Supplemental tags can provide additional information and may be useful for inventory, but the discovery tool should not use the tag identifier from supplemental tags to reference the product itself (cf. **GEN-17**). If the tag identifier already exists in inventory, the discovery tool will determine if the tag version has changed by examining the value associated with the `<SoftwareIdentity>` element's `@tagVersion` attribute. If that tag version has been updated, the tool will register the updated values that were changed in the SWID tag (cf. **GEN-18**, **GEN-19**).

8. The local repository will be updated, including sending notifications 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.

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 discovery 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 and may be supplemented with a periodic full refresh. This provides for efficiencies in the velocity and volume of information that needs to be exchanged.

9. For a given endpoint, the discovery tool iterates through each tag in the repository, including non-authoritative SWID tags.

10. The endpoint-collected tags are added to the enterprise repository, recording relevant endpoint identification information (host name, IP addresses, etc.), the date and time of the data collection, and data about the discovery tool or remote management interface used.

11. The discovery tool will record relationships between tags, as indicated within the SWID tags discovered. For example, patch tags include a reference (using the `<Link>` element's `@href` and `@rel` attributes) to the software being modified (cf. **PAT-1**). Similarly, for supplemental tags recorded, the discovery tool will indicate the tag identifier for the primary tag of the software for which additional information is being provided (cf. **SUP-1**).

5.1.1.3 Outcomes

This combination of activities provides an accurate and automated method 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 SWID tags rather than attempting to infer inventory information by examining arbitrary indicators on the endpoint (e.g., registry keys, installed files).

SWID tags contribute to a reliable SAM inventory by supporting 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 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 data identified through the `<Link>` and `<Meta>` elements. Details regarding attributes and values that can be useful for queries are described in Section 2.3.

In many cases, the ability to consistently 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. The practitioner is able to know what is installed and where it is installed, providing a critical foundation for other usage scenarios.

5.1.2 US 2 - Ensuring that Products Are Properly Patched

Consumers often need reports about endpoints that are missing a patch, for security awareness or management purposes. If a discovery tool also has a patch management capability, it will need to determine that all prerequisite patches are installed before installing any new patches. While this usage scenario focuses on an enterprise patch management approach, a local patch management capability that is executed on an endpoint can directly read the inventory of patch tags from the local repository to enable localized patch verification and decision making.

5.1.2.1 Initial Conditions

This usage scenario assumes the existence of an enterprise repository, populated with SWID tags that are created and collected using the process described in US 1 (Section 5.1.1). This includes application of guidelines **GEN-1** through **GEN-4**.

5.1.2.2 Process

1. Through a dashboard or other internal process, the discovery tool determines that a given software product needs to be patched (e.g., for a functional update, due to a discovered vulnerability).

2. If the tag identifier of the required patch is known, the discovery tool searches through the patch tags recorded in the repository for records indicating that a patch tag with the designated identifier is installed on an endpoint. If the patch tag identifier is unknown, the discovery tool will search for patch tags with a name that matches the `<SoftwareIdentity>` `@name` of the desired patch.

3. The discovery tool then examines the patch tag to determine if it includes any other required predecessor patches (cf. **PAT-3**). If there is no such requirement, or if the required patches are also confirmed as installed on the endpoint, the endpoint is recorded as properly patched for this instance.

4. If desired, the discovery tool can validate each file expected to be added, modified, or removed by the given patch(es). Patch tags created in accordance with Section 4.3.3 (cf. **PAT-5, PAT-6, PAT-7**) clearly specify the files that are modified as a result of applying the patch. The discovery tool enumerates each of the files shown as added or modified within the `<Payload>` element of a patch tag as indicated by the `@patchEvent` attribute. The tool compares the recorded filename and cryptographic hash with the actual files that reside on the endpoint. The discovery tool can also confirm deletion of those files that the patch tag indicates should have been removed.

5. Where a patch is noted as missing, the discovery tool can take advantage of relationships to other patches, as described in Section 2.1.3, to see if that patch has been superseded by a newer patch (cf. **PAT-4**). In this case, the discovery tool can examine known patch tags for any that are known to supersede the desired patch, noting that the former patch may no longer apply.

6. The search results are provided through the discovery tool's dashboard and/or reporting process.

5.1.2.3 Outcomes

The discovery tool user is able to accurately and quickly identify instances where a required patch or update is not installed on a given endpoint. If patched files are also assessed, the user is able to verify patch installations. The user is able to determine which endpoints meet (or do not meet) specific patch requirements, supporting security situational awareness and patch/vulnerability management as part of a continuous monitoring solution.

5.1.3 US 3 - Correlating Inventory Data with Vulnerability Data to Identify Vulnerable Endpoints

The remediation of known vulnerabilities through timely patching is considered a vulnerability management best practice. SWID tags improve vulnerability management by providing comprehensive, compact descriptions of installed software and patches, which may then be compared and correlated with vulnerability information. Through the knowledge gained by examining SWID tags, further automation of vulnerability management is possible.

Because of their use of a consistent and standardized structure, SWID tags promote effective correlation of information published by vulnerability information sources (e.g., NVD, vendor alerts, US-CERT alerts) with the inventory information collected by discovery tools. Many vulnerability bulletins use the CPE specification to identify classes of products that are affected by a vulnerability [CPE23N]. Appendix A describes a method to form CPE names automatically from input SWID tags. This capability can be used to translate a software inventory based on SWID tags to one based on CPE names. Given a vulnerability bulletin that references products using CPE names, this translation can then be used to identify potentially vulnerable endpoints.

If a tag creator uses the appropriate `<Meta>` attributes to specify additional detailed naming information in a product’s primary tag (cf. Error! Reference source not found.), this information becomes readily available to publishers of vulnerability bulletins. By including appropriate references to those attribute values, bulletins make it easier for consumers to accurately search SWID-based inventory data for affected products. For example, if the presence or absence of a product vulnerability depends on software edition information, it is advantageous both for tag creators to specify the `<Meta> @edition` attribute, and for publishers of vulnerability bulletins to reference that value explicitly.

5.1.3.1 Initial Conditions

This usage scenario assumes the existence of an enterprise repository populated with SWID tags that are created and collected using the process described in US 1 (Section 5.1.1). This includes application of guidelines **GEN-1** through **GEN-4**.

5.1.3.2 Process

1. Using software vulnerability bulletins containing information about reported software vulnerabilities, the discovery tool searches for endpoints on which the referenced software is installed. The search criteria may include SWID tag information such as the information provided in the `<SoftwareIdentity> @name` and `@version` (cf. **PRI-2**), and `<Meta> @revision` and `@edition` (cf. **PRI-13,** Section 4.2.4). By forming CPE names from SWID tags (cf. Appendix A) based on the collected SWID-based software inventory, those CPE names can be used to search for related vulnerability bulletins.

2. If the bulletin references the tag identifier for the relevant tag for a software product or patch, this will help provide a rapid and accurate query of related installations (cf. **GEN-15**). Often, this information will only refer to a given software product name and version; SWID tags

adhering to guidelines **PRI-1** through **PRI-6** enable the discovery tool to automatically and accurately correlate inventory and vulnerability data.

3. If the bulletin only references one or more known filename(s), but does not identify the software product itself, it will be necessary to search for software products and patches that include the file(s). Guidelines **PRI-8** through **PRI-12** ensure that filename information is captured in the `<Payload>` and/or `<Evidence>` elements of SWID tags to support this type of query. As a result, the discovery tool can search the `<Payload>` and/or `<Evidence>` portions of recorded tag information in the repository to look for software and patches of interest.

   For example, to identify instances of the "Heartbleed bug", the tool might search for any tags where the `<Payload>` and/or `<Evidence>` portions of recorded tags contain references to the vulnerable OpenSSL library. Products including this library can be identified and then those products can be searched for to identify vulnerable software installations.

4. Where a record exists that matches the query parameters, as described above, the associated endpoint is flagged as containing vulnerable software.

5. Where patch tag information is provided in the bulletin, the discovery tool queries the repository to determine whether the appropriate patch tag has been installed (cf. US 2, Section 5.1.2), including checks for predecessor or superseded patches.

6. If the endpoint is found to contain vulnerable software but not the associated patch(es), the system may be flagged as requiring potential remediation activities.

Consider the case of a fictional vulnerability involving a known buffer overflow in the product named Acme Roadrunner, affecting versions between 11.1 and 12.1 (inclusive). The issue is remediated in version 12.2 and later. There is also a patch KB123 that remediates the vulnerability. The discovery tool can 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 discovery tool searches through the repository and discovers a patch tag named "Acme_Roadrunner_KB123" associated with that endpoint.

Given the above scenario, the discovery 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, also supporting security analysis.

5.1.3.3 Outcomes

Through the use of SWID tags for the description and discovery of vulnerable software, organizations are able to identify known vulnerabilities within their enterprise based on SWID-based software inventory data and published vulnerability bulletins.

5.1.4 US 4 - Discovering Vulnerabilities Due to Orphaned Software Components

Orphaned software is a component of a previously installed software product that has since been uninstalled. Examples include shared library files and software patches. If these orphaned components contain an exploitable flaw, their presence can expose an endpoint to additional security risk. Additionally, orphaned software may waste resources on an endpoint.

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 (cf. Section 2.3.4) enables understanding of how software components relate, supporting software asset management decisions.

5.1.4.1 Initial Conditions

This usage scenario assumes the following conditions:

- Existence of an enterprise repository, populated with SWID tags that are created and collected using the process described in US 1 (Section 5.1.1). This includes application of guidelines **GEN-1** through **GEN-19**.
- 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 2.3.4.

5.1.4.2 Process

1. For a given endpoint (or set of endpoints), the discovery tool iterates through each tag in the repository, including non-authoritative SWID tags. The tool specifically inspects tags that indicate a relationship to other products as indicated by the `<Link>` element's `@rel` attribute with a value of "`parent`". Such tags will include an `@href` pointer to the parent software component. Users may also find tags that point to software that requires subsidiary software components (e.g., patches, C/C++ runtime libraries) using a `@rel` value of "`requires`".
2. For each tag selected, the discovery tool searches for the existence of SWID tags that indicate a relationship to a SWID tag for a parent software product, to confirm installation of that parent software product. Where a match is <u>not</u> located, the discovery tool records that an orphaned software component might exist on that endpoint.
3. Where the discovery tool has information about a subsidiary product that is required by another product (as indicated by the `@rel` "`requires`" value), the tool can confirm that there are still SWID tags on that endpoint that require the subsidiary product. For example, if

Product A originally required a runtime library, the discovery tool could periodically confirm that Product A remains installed and still requires that library. If Product A is removed and no other software products require the library, that library should likely be removed as well.

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

5.1.4.3 Outcomes

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

5.2 Enforcing Organizational Software Policies

This cybersecurity objective area focuses on the use of SWID tags to help security practitioners minimize security risk by enforcing enterprise policies regarding authorized software. These policies may be implemented as blacklists (lists of prohibited products, with all unlisted products implicitly allowed) or whitelists (lists of allowed products, with all others prohibited). In addition, specific products may be designated as mandatory by the enterprise (e.g., antivirus and intrusion detection and prevention applications), possibly conditionalized on endpoint role (e.g., end-user workstations versus Internet-facing web servers). Policies may be enforced at time of (attempted) product installation, and/or any time thereafter.

This section describes the following usage scenarios:

- US 5 – Preventing Installation of Unauthorized or Corrupted Software Products (Section 5.2.1)
- US 6 – Discovering Corrupted Software and Preventing Its Execution (Section 5.2.2)
- US 7 – Preventing Potentially Vulnerable Endpoints from Connecting to Network Resources (Section 5.3)

5.2.1 US 5 - Preventing Installation of Unauthorized or Corrupted Software Products

To strictly control what software may or may not be installed on information systems, discovery tools supported by corpus tag information can ensure that all installed software on a given endpoint matches the specification of a whitelist, or does not match the specification of a 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 yet another. Similarly, endpoints that are dedicated to a particular role (e.g., "messaging server") might have unique lists of allowed or prohibited software.

As described in Section 2.1.1, corpus tags may be used to authenticate the issuer of a software product installation package 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 2.3.6 on the `<Payload>` element), the installation routine can confirm (from the whitelist) that the software's integrity has been preserved, preventing unauthorized modifications to software distributions.

5.2.1.1 Initial Conditions

This usage scenario assumes that the following conditions exist:

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

5.2.1.2 Process

1. Upon execution of a software installation, the installation tool discovers a corpus tag included with the software distribution. Installers may be configured to prevent installation of software that does not supply a corpus tag, ending this process.
2. If provided, the installation tool may examine the value of the `@thumbprint` attribute(s) of the `<Entity>` element of the signer, comparing the hexadecimal string that contains a hash of the signer's certificate. This allows each digital signature to be directly related to the entity specified. The installation tool validates the signer's certificate and the tag's signature if the corpus tag is signed. If the signature is found to be invalid, the installation is prevented, ending this process.
3. The installation tool determines whether the `@tagId` or other data included in the tag (cf. **COR-1** through **COR-6)** matches the criteria specified in either a whitelist or a blacklist. If the evaluation of the tag is determined to be in violation of the whitelist or blacklist policy, then installation is prevented, ending this process.
4. The installation tool verifies the cryptographic hashes of the installation files using the `<Payload>` data included in the corpus tag (cf. **COR-6**). If any hash is found not to match, the installation is prevented, ending this process.

While steps 1 and 2 can be independent policy decisions, steps 3 and 4 rely on the assurance provided by validating the tag's signature in step 2. Thus, steps 3 and 4 cannot be conducted if step 2 is omitted by policy.

5.2.1.3 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.

5.2.2 US 6 - Discovering Corrupted Software and Preventing Its Execution

Similar to US 5 described above, effective software asset management includes the ability to discover potentially tampered software that has already been installed on an endpoint. This is accomplished by comparing the known hash values of installed software/packages (as recorded in the local endpoint repository and/or the enterprise repository) to the files observed on the endpoint.

Detection of potential tampering may be used for several purposes, including the following:

- To report potential compromise of an endpoint
- To quarantine an endpoint pending further investigation
- 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 3.4 will help provide confidence in the findings. Comparison of observed hash values with recorded values in the enterprise repository requires additional network and computing resources, and is more commonly performed as a periodic monitoring task.

5.2.2.1 Initial Conditions

This usage scenario assumes the existence of an enterprise repository populated with SWID tags that are created and collected using the process described in US 1 (Section 5.1.1). This includes application of guidelines **GEN-1** through **GEN-13**, **GEN-18**, and **GEN-19**.

5.2.2.2 Process

1. For each endpoint, the discovery tool reads each primary and supplemental SWID tag, examining the stored cryptographic hashes for each file listed in the `<Payload>` or `<Evidence>` elements. This information will have been collected and included in the SWID tag, for example, as described in guidelines **PRI-8** through **PRI-12,** to provide detailed information about the files comprising an installed software product. Detailed information regarding the creation and comparison of cryptographic hashes with sufficient security strength is described in Section 3.4.

   Similarly, the discovery tool examines each patch tag to gather the cryptographic evidence for files added or changed by that patch (cf. **PAT-5** through **PAT-7)**.

2. The discovery 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 calculated file hash does not match the one provided in the SWID tag, the discovery tool will report the variance.

4. The tool may also, based upon the detection of potential tampering, prevent execution of that software product.

5.2.2.3 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 otherwise altered maliciously.

5.3 US 7 - Preventing Potentially Vulnerable Endpoints from Connecting to Network Resources

A forward-looking approach to improving organizations' cybersecurity is to prevent potentially vulnerable endpoints from connecting to the network, or to move such endpoints to an isolated network segment for remediation or investigation. Currently, products are available that achieve this through proprietary methods. Additionally, groups are working on open standards to accomplish this goal. For example, the Trusted Computing Group's Trusted Network Connect Working Group (TNC-WG) has defined an open solution architecture that enables network operators to enforce policies regarding the security state of endpoints in order to determine whether to grant access to a requested network infrastructure. The use of SWID tags provides a technology-neutral way to verify an endpoint's compliance with certain configuration policies (e.g., updated antivirus definitions, configuration compliance with baseline specifications) and safeguards against known software vulnerabilities (e.g., updated patches).

5.3.1.1 Initial Conditions

This usage scenario assumes the existence of an enterprise repository populated with SWID tags that are created and collected using the process described in US 1 (Section 5.1.1). This includes application of guidelines **GEN-1** through **GEN-13**, **GEN-18**, and **GEN-19**.

5.3.1.2 Process

1. An endpoint attempts to access a given network resource, such as an enterprise wide area network (WAN) or a protected website.

2. Using information from the local SWID tag repository, a client on the endpoint collects the information needed to support a network access decision. Examples of the information to be collected include:

    - The inventory of installed software and patches

    - File metadata, including names, hashes, sizes, and versions

3. The client securely transfers this information to a policy decision point.

4. The policy decision point renders a decision based on the provided data, and issues a network access recommendation to a policy enforcement point (e.g., a network switch). The policy enforcement point might allow the endpoint on the network, place it on a restricted network, or quarantine the endpoint in order to remedy a potential risk (e.g., by updating patches or

antivirus definitions). If an endpoint is assessed to be in violation of policy, but not to present a significant risk, it may be allowed on the network, but subjected to detailed monitoring.

SWID-related data provided to a policy decision point may be used to determine:

- If any known software vulnerabilities exist on the endpoint (cf. Section 5.1.1)
- If an endpoint is properly patched (cf. Section 5.1.2)
- If the endpoint is in compliance with whitelist or blacklist requirements (cf. Section 5.2.1)

5.3.1.3 Outcomes

Through the use of well-formed SWID tags, network access decision points are able to collect validation information quickly and accurately, enabling the organization to help prevent the connection of endpoints that represent a potential threat.

5.4 Association of Usage Scenarios with Guidelines

To aid in the association of usage scenarios to the guidelines described in Sections 3 and 4, the guidelines are organized into sets of families, described below, which help aggregate items into relevant groupings. These items follow the coded identifier format described in the introduction to Section 3 and are grouped into the following categories:

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

The guideline families are:

- **Basic Compliance** – Tags must be well-formed in accordance with the **2015** revision of the ISO 19770-2 specification so that they are interoperable among discovery tools (cf. **GEN-1**).
- **Tag Type** – Discovery tools need to be able to determine the type of SWID tag that is discovered or recorded. Corpus, patch, and supplemental tags are readily identified (cf. **COR-1, PRI-1, PAT-1, SUP-1**) to aid in achieving SAM objectives and attaining cybersecurity goals.
- **Tag Authority** – Discovery tools need to be able to inspect a SWID tag and rapidly determine the authority of the tag creator, since authoritative tags have priority and may be more accurate than non-authoritative tags (cf. **GEN-2** through **GEN-4**).

- **File Information** – Discovery tools use the files' names, sizes, and cryptographic hashes included in SWID tags to validate integrity (cf. **GEN-5** through **GEN-12)**.

- **XML Signature –** Digital signatures, where they can be used, help to authenticate and verify the integrity of a SWID tag (cf. **GEN-13**).

- **Tag Identification** – Proper use of the `<SoftwareIdentity>` element's `@tagId` attribute helps discovery tools refer to product installation packages, releases, and patches (cf. **GEN-14** through **GEN-17)**.

- **Tag Updates** – While SWID tags are rarely changed, when this occurs (e.g., to correct an error) the tag identifier stays the same but the tag version needs to be incremented to note that there is updated SWID tag information available (cf. **GEN-18, GEN-19)**.

- **Pre-Installation Software Version** – It is important that the version of a software product on pre-installation media is provided, along with the version scheme (e.g., 1.2.3 or Version A, Version B) so that discovery tools can quickly and accurately understand the product to be installed. Proper use of well-known version schemes helps ensure that software installation versions are consistently referenced (cf. **COR-2** through **COR-4**).

- **Installation Media Integrity Verification –** Software installation products and discovery tools can use the payload information to prevent installation of software from media that might be corrupted (cf. **COR-6)**.

- **Post-Installation Software Version** – Many of the cybersecurity objectives depend upon the ability to accurately identify both the name and version of software products installed. Application of primary tag guidelines supports identification through SWID tags that are interoperable and consistent (cf. **PRI-2** through **PRI-6)**.

- **Software Integrity Verification** - Detailed information about the files comprising an installed software product is critical to help confirm that the correct files are installed and to verify the integrity of those files. This information is derived from the `<Payload>` and/or `<Evidence>` information included in the SWID tag (cf. **PRI-8** through **PRI-12)**.

- **Metadata** – Software products are often known by colloquial identifiers as well as by formal version references. Where metadata applies to the software identified by a SWID tag, it MUST be included in the tag to help accurately identify products and components installed (cf. Error! Reference source not found.).

- **Patch Relationships** – By their nature, software patches are related to one or more software products. They may also require or supersede other software or patches. To ensure consistent and accurate SAM reporting, a SWID patch tag needs to comply with guidelines **PAT-2** through **PAT-4**.

- **Patch Integrity Verification** – As with software integrity, detailed information about the files contained within a software patch is also critical to help confirm patch integrity. To manage the software assets properly, it is also critical to understand what files are modified, added, or removed. This information is derived from the information included in the SWID patch tag (cf. **PAT-5** through **PAT-7)**.

- **Supplemental Tag Relationships** – Supplemental SWID tags, by definition, provide additional information that supplements the data in another tag. Because the SWID specification does not clearly state how a supplemental tag should indicate its linkage to such other tags, it is important to follow guideline **SUP-2** to ensure consistent and accurate understanding of the relationships among tags.

- **Data Deconfliction** - Supplemental tags are secondary to the primary, corpus, and patch tags they support; guideline **SUP-3** helps ensure that the original tag (the one being supplemented) has the correct and primary information in case of any conflict.

Table 3 illustrates the association among the usage scenarios and the previously described guidelines. The usage scenarios demonstrate the rationale for each of the guidelines to help organizations consistently achieve important cybersecurity objectives through the appropriate creation and usage of well-formed SWID tags.

Table 3: Relationship of Guidelines to Usage Scenarios

Guideline Family	Guideline	Usage Scenarios						
		US1	US2	US3	US4	US5	US6	US7
Basic Compliance	GEN-1	●	●	●	●	●	●	●
Tag Authority	GEN-2[A]	●	●	●	●		●	●
	GEN-3[A]	●	●	●	●		●	●
	GEN-4[N]	◒	◒	◒	◒		◒	◒
File Information	GEN-5	●			●		●	●
	GEN-6	●			●		●	●
	GEN-7[A]	●			●		●	●
	GEN-8[N]	◒			◒		◒	◒
	GEN-9	◒			◒		◒	◒
	GEN-10[A]	●			●		●	●
	GEN-11[N]	◒			◒		◒	◒
	GEN-12	◒			◒		◒	◒
XML Signature	GEN-13	◒			◒		◒	◒
Tag Identification	GEN-14	◒			◒			
	GEN-15	◒		◒	◒			
	GEN-16	◒		◒	◒			
	GEN-17	◒			◒			
Tag Updates	GEN-18	◒			◒		◒	◒
	GEN-19	●			●		●	●
Tag Type	COR-1	●				●		
	PRI-1	●						

	PAT-1	●						
	SUP-1	●						
Pre-Installation Software Version	COR-2					●		
	COR-3					●		
	COR-4					◒		
	COR-5					◒		
Installation Media Integrity Verification	COR-6					●		
Post-Installation Software Version	PRI-2[A]	●		●				
	PRI-3[A]	●		●				
	PRI-4[N]			◒				
	PRI-5[N]			◒				
	PRI-6	◒		◒				
	PRI-7	◒		◒				
Software Integrity Verification	PRI-8[A]			●			●	
	PRI-9[N]			◒			◒	
	PRI-10			◒			◒	
	PRI-11[A]			●			●	
	PRI-12[N]			●			●	
Metadata	PRI-13			●				
Patch Relationships	PAT-2		●					
	PAT-3[A]		●					
	PAT-4[A]		●					
Patch Integrity Verification	PAT-5[A]		●				●	
	PAT-6[A]		●				●	
	PAT-7[N]		●				●	
Supplemental Tag Relationships	SUP-2							
Data Deconfliction	SUP-3							
● indicates mandatory guideline, ◒ indicates recommended								

Appendix A—Forming Common Platform Enumeration (CPE) Names

A component of NIST's Security Content Automation Protocol (SCAP), the Common Platform Enumeration (CPE) is a standardized method of naming classes of applications, operating systems, and hardware devices that may be present on computing devices.[5] NIST maintains a dictionary of CPE names as part of the National Vulnerability Database (NVD).[6] CPE names play an important role in the NVD, where they are used to associate vulnerability reports with the affected software products. Many cybersecurity products report discovered software using CPE names, and/or use CPE names to search the NVD for indications of software vulnerability. For these reasons, it is useful to specify a standardized, automatic procedure for forming CPE names using pertinent SWID tag attribute values. This appendix defines such a procedure.

A.1 CPE Name Forming Challenges and Solutions

The CPE Name Forming Procedure presented here conforms to version 2.3 of the CPE Naming Specification [CPE23N]. This specification defines eleven attributes comprising a well-formed CPE name (WFN):

- part
- vendor
- product
- version
- update
- edition
- language
- sw_edition
- target_sw
- target_hw
- other

Two challenges must be addressed when forming a CPE name automatically from data contained in a SWID tag. The first challenge is *data insufficiency*, and the second is *non-ASCII characters*. These are discussed in the following subsections.

A.1.1 Data Insufficiency

A SWID tag that conforms only to the mandates and requirements set forth in the SWID specification would lack the data required to reliably populate nine of the eleven attributes of a

---

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

CPE name. One could use <SoftwareIdentity> @name as the value for the CPE "product" attribute, and <SoftwareIdentity> @version as the CPE "version" attribute, but the other CPE attributes have no obvious sources within a SWID tag and thus would have to be left unspecified in any automatically generated CPE name.

Unfortunately, a CPE name that includes only a product name and a version will, in most cases, be insufficient for vulnerability management usage scenarios. In particular, using such a limited CPE name to search the NVD for vulnerability reports is likely to result in a *false negative*: a failure to discover relevant software vulnerability reports in the NVD even when such relevant reports exist. False negatives are likely because the SWID specification supplies only the <SoftwareIdentity> @name attribute to capture a product's market name, whereas the CPE specification breaks a product's name down into a set of fine-grained data elements, including *vendor*, *product*, *update*, *edition*/*sw_edition*, and *hw_edition*.

Consider a product with the market name assigned by the vendor of "Acme Roadrunner Home Edition Service Pack 2." This is the string that would be specified as the value of the product's <SoftwareIdentity> @name attribute in its primary tag. In contrast, a conventional CPE name as used within the NVD would break that string into the following CPE name elements:

vendor = "acme"
product = "roadrunner"
update = "sp2"
sw_edition = "home"

As a result, vulnerability reports in the NVD associated with "Acme Roadrunner Home Edition Service Pack 2" would be tagged with the following CPE standard-conformant name:

```
cpe:2.3:acme:roadrunner:*:sp2:*:*:*:home:*:*:*
```

Now consider attempting to generate a CPE name from the Acme Roadrunner product's primary SWID tag. A name generation procedure that used only the tag's <SoftwareIdentity> @name and @version attributes would produce the following CPE name (assuming straightforward replacement of whitespace with underscores, and character conversion to lowercase):

```
cpe:2.3:*:acme_roadrunner_home_edition_service_pack_2:*:*:*
:*:*:*:*:*:*
```

A search of the NVD using this generated CPE name—applying the matching algorithm that is defined as part of the CPE specification—would likely fail to find any records, including those records tagged with the standard-conformant name. This negative result would create the false impression that the Acme Roadrunner product is free of known vulnerabilities.

Guideline **PRI-13** in Section 4.2.4 of this report requires that several additional data values be provided in SWID tags, using the <Meta> element:

- `@product`
- `@colloquialVersion`
- `@revision`
- `@edition`

In addition, guideline **GEN-3** in Section 3.3 of this report requires authoritative tag creators to specify an `<Entity>` `@name` for the `softwareCreator` role, and guideline **GEN-4** encourages non-authoritative tag creators to do so whenever possible. These guidelines make it possible to form more useful CPE names from a SWID tag.

A.1.2 Non-ASCII Characters

CPE names are limited to the printable subset of the American Standard Code for Information Interchange (ASCII) character encoding set. In contrast, when strings are used as SWID tag attribute values, those strings may contain arbitrary Unicode characters. This creates a need for a standard approach for converting Unicode characters into ASCII characters acceptable within a CPE name.

IETF RFC 3490 on Internationalizing Domain Names in Applications (IDNA) [RFC 3490] offers a solution to this challenge. IDNA defines the concept of an ASCII-Compatible Encoding (ACE) of a string, which may contain arbitrary ASCII and non-ASCII characters, and further specifies a `ToASCII` procedure that converts such strings into strings composed of only ASCII characters. Although the output of `ToASCII` is not intended for human consumption, it provides a satisfactory encoding of the input that meets the requirements for CPE name attributes. In addition, IDNA also offers a `ToUnicode` procedure that takes an ACE string as input and reverses the encoding to produce an output string, which may contain Unicode characters. Consequently, guidance in this appendix will require that pertinent SWID tag attribute value strings are processed by an RFC 3490-conformant implementation of `ToASCII` during the CPE name forming procedure.

A.2 Overview of CPE Name Forming Procedure

The `CPENameGenerator` procedure, formally specified in Appendix A.3 below, has the following steps:

1. Given an input SWID tag, a collection of *preliminary CPE name attributes* is extracted. These attributes are "preliminary" in the sense that their values are directly copied from the input tag and do not yet conform to the CPE attribute requirements (e.g., containing only printable ASCII characters).

2. Each preliminary CPE name attribute is converted to the ASCII encoding using the `ToASCII` procedure specified in [RFC 3490].

3. Any embedded whitespace characters are replaced with underscore characters.

4. Printable non-alphanumeric characters *except underscores* are quoted (i.e., a backslash character is inserted into the string immediately before the non-alphanumeric character.

5. Values for the final CPE name attributes are assigned. In most cases, final values are simply the results of the preceding four steps. Special conditions apply to how the CPE "product" value is assigned.

The CPENameGenerator produces a CPE WFN as its output. This WFN may then be bound to either a URI or a formatted string according to the bind_to_URI() and bind_to_fs() procedures specified in [CPE23N].

A.3 CPENameGenerator Procedure in Detail

The CPENameGenerator procedure is formally specified below.

A.3.1 Step 1 – Collect Preliminary CPE Name Attributes

Given an input SWID tag, extract the following preliminary attribute values:

prelimVendor := value of <Entity> @name where <Entity> @role contains softwareCreator

prelimProduct := value of <Meta> @product

prelimProductDefault := value of <SoftwareIdentity> @name

prelimColloqVer := value of <Meta> @colloquialVersion

prelimVersion := value of <SoftwareIdentity> @version

prelimUpdate := value of <Meta> @revision

prelimEdition := value of <Meta> @edition

A.3.2 Step 2 – Convert to Pure ASCII

The ToASCII procedure is applied to each preliminary attribute value:

```
prelimVendor := ToASCII(prelimVendor)

prelimProduct := ToASCII(prelimProduct)

prelimProductDefault := ToASCII(prelimProductDefault)

prelimColloqVer := ToASCII(prelimColloqVer)

prelimVersion := ToASCII(prelimVersion)

prelimUpdate := ToASCII(prelimUpdate)
```

```
    prelimEdition := ToASCII(prelimEdition)
```

A.3.3 Step 3 – Replace Whitespace with Underscores

Apply the `eliminate_whitespace()` function to each preliminary attribute value:

```
    prelimVendor := eliminate_whitespace(prelimVendor)

    prelimProduct := eliminate_whitespace(prelimProduct)

    prelimProductDefault :=
      eliminate_whitespace(prelimProductDefault)

    prelimColloqVer := eliminate_whitespace(prelimColloqVer)

    prelimVersion := eliminate_whitespace(prelimVersion)

    prelimUpdate := eliminate_whitespace(prelimUpdate)

    prelimEdition := eliminate_whitespace(prelimEdition)
```

The `eliminate_whitespace()` function is defined as follows:

```
function eliminate_whitespace(s)
  ;; Inspect each character in string s. In the output, replace
  ;; any embedded whitespace characters with underscores.
  result := "".
  idx := 0.

  while (idx < strlen(s))
    do
      c := substr(s,idx,idx). ; get the idx’th character of s.
      if is_whitespace(c) then
        ;; Substitute an underscore for a whitespace character.
        result := strcat(result,”_”).
      else
        result := strcat(result,c).
      endif.
      idx := idx + 1.
  end.
  return result.
end.

function substr(s,b,e)
  ;; Returns a substring of s, beginning at the b’th character,
  ;; with 0 being the first character, and ending at the e’th
  ;; character. b must be <= e. Returns nil if b >= strlen(s).
end.
```

```
function strcat(s1,s2,...sn)
  ;; Returns a copy of the string s1 with the strings s2 to sn
  ;; appended in the order given.
  ;; Cf. the GNU C definition of strcat. This function shown
  ;; here differs only in that it can take a variable number
  ;; of arguments. This is really just shorthand for
  ;; strcat(s1, strcat(s2, strcat(s3, … ))).
end.

function strlen(s)
  ;; Defined as in GNU C, returns the length of string s.
  ;; Returns zero if the string is empty.
end.
```

A.3.4 Step 4 – Add Quoting as Required

Apply the `add_quoting()` function to each preliminary attribute value:

```
prelimVendor := add_quoting(prelimVendor)

prelimProduct := add_quoting(prelimProduct)

prelimProductDefault := add_quoting(prelimProductDefault)

prelimColloqVer := add_quoting(prelimColloqVer)

prelimVersion := add_quoting(prelimVersion)

prelimUpdate := add_quoting(prelimUpdate)

prelimEdition := add_quoting(prelimEdition)
```

The `add_quoting()` function is defined as follows:

```
function add_quoting(s)
  ;; Inspect each character in string s. Alphanumeric characters
  ;; and underscores pass unchanged. All other characters are
  ;; prefixed with a backslash (quote) character.
  result := "".
  idx := 0.

  while (idx < strlen(s))
    do
      c := substr(s,idx,idx). ; get the idx'th character of s.
      if (is_alphanum(c) or c = "_") then
        ;; Alphanumerics and underscores pass untouched.
        result := strcat(result,c).
      else
```

```
        result := strcat(result,"\").
        result := strcat(result,c).
      endif.
      idx := idx + 1.
  end.
end.
```

A.3.5 Step 5 – Finalize the CPE WFN Attribute Values

The final CPE WFN attribute values are assigned as follows:

`part` := "*"

`vendor` := `prelimVendor` (if non-null) otherwise "*"

`product` := `prelimProduct` (if non-null) otherwise `prelimProductDefault`
In addition, if `prelimColloqVer` is non-null, then add it to the `product` attribute:
`product := product + "_" + prelimColloqVer`

`version := prelimVersion`

`update := prelimUpdate` (if non-null) otherwise "*"

`edition := prelimEdition` (if non-null) otherwise "*"

all other WFN attributes := "*"

The resulting eleven attribute values now satisfy the requirements of a CPE WFN and are suitable for binding to URI or formatted string names.

A.4 Guidelines on CPE Name Formation

This appendix concludes with guidelines related to the formation of CPE names from SWID tags.

The first guideline limits the applicability of CPE Name Formation to only two types of SWID tags: corpus and primary tags. Because corpus tags are used to describe software products in a pre-installation state, it is useful to be able to form CPE names from such tags in cases where CPE name information could be helpful in deciding, for example, whether or not to allow installation. Because primary tags describe software products installed on endpoints, it is useful to be able to form CPE names from such tags to support vulnerability management usage scenarios. Because CPE was never designed to support naming of patches, patch tags cannot be used as sources for CPE names. Supplemental tags are not useful as sources of CPE names since only corpus and primary tags may contain the necessary data values.

Guidelines on CPE name formation are provided as additions to the tag-specific implementation guidelines set forth in Section 4:

COR-7. A corpus tag MAY be used as the source of a CPE name. When forming a CPE name from a corpus tag, the `CPENameGenerator` procedure MUST be followed.

PRI-14. A primary tag MAY be used as the source of a CPE name. When forming a CPE name from a primary tag, the `CPENameGenerator` procedure MUST be followed.

PAT-8. A patch tag MUST NOT be used as the source of a CPE name.

SUP-4. A supplemental tag MUST NOT be used as the source of a CPE name.

Appendix B—Acronyms

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

ABNF	Augmented Backus-Naur Form
ACE	ASCII-Compatible Encoding
API	Application Programming Interface
ASCII	American Standard Code for Information Interchange
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
IANA	Internet Assigned Numbers Authority
ID	Identifier
IDNA	Internationalizing Domain Names in Applications
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
ISSO	Information System Security Officer
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 Comments
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
TNC-WG	Trusted Network Connect Working Group
URI	Uniform Resource Identifier
US	United States
US	Usage Scenario

USB	Universal Serial Bus
US-CERT	United States Computer Emergency Readiness Team
USG	United States Government
W3C	World Wide Web Consortium
WAN	Wide Area Network
WFN	Well-Formed CPE Name
XAdES	XML Advanced Electronic Signature
XAdES-T	XML Advanced Electronic Signature with Time-Stamp
XML	Extensible Markup Language
XPath	XML Path Language
XSD	XML Schema Definition

Appendix C—References

[CPE23N] Cheikes, B. A., Waltermire, D., and Scarfone, K. *Common Platform Enumeration: Naming Specification 2.3*. National Institute of Standards and Technology Interagency Report 7695, August 2011. http://csrc.nist.gov/publications/nistir/ir7695/NISTIR-7695-CPE-Naming.pdf [accessed 8/26/15].

[FIPS180-4] U.S. Department of Commerce. *Secure Hash Standard (SHS)*, Federal Information Processing Standards (FIPS) Publication 180-4, August 2015. http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.180-4.pdf [accessed 8/26/15].

[FIPS186-4] U.S. Department of Commerce. *Digital Signature Standard (DSS)*, Federal Information Processing Standards (FIPS) Publication 186-4, July 2013. http://dx.doi.org/10.6028/NIST.FIPS.186-4 [accessed 8/26/15].

[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 8/26/15].

[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 8/26/15].

[NISTIR 7802] Booth, H., and Halbardier, A. (2011). *Trust Model for Security Automation Data 1.0.* National Institute of Standards and Technology Interagency Report 7802, 2011. http://csrc.nist.gov/publications/nistir/ir7802/NISTIR-7802.pdf [accessed 8/26/15].

[RFC 3490] Faltstrom, P., Hoffman, P., and Costello, A. (2003). *Internationalizing Domain Names in Applications (IDNA).* Internet Engineering Task Force (IETF) Network Working Group Request for Comments (RFC) 3490, March 2003. https://www.ietf.org/rfc/rfc3490.txt [accessed 8/26/15].

[RFC 3986] Berners-Lee, T., Fielding, R., and Masinter, L. *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 8/26/15].

[RFC 5234] Crocker, D., and Overell, P. *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 8/26/15].

[SEMVER] Preston-Werner, T. *Semantic Versioning 2.0.0.* http://semver.org/spec/v2.0.0.html [accessed 8/26/15].

[SP800-107] Dang, Q. *Recommendation for Applications Using Approved Hash Algorithms*, NIST Special Publication (SP) 800-107 Revision 1, National Institute of Standards and Technology, August 2012. http://csrc.nist.gov/publications/nistpubs/800-107-rev1/sp800-107-rev1.pdf [accessed 8/26/15].

[SP800-57-part-1] Barker, E. et al. *Recommendation for Key Management – Part 1: General*, NIST Special Publication (SP) 800-57 Part 1 Revision 3, National Institute of Standards and Technology, July 2012. http://csrc.nist.gov/publications/nistpubs/800-57/sp800-57_part1_rev3_general.pdf [accessed 8/26/15].

[XAdES] Cruellas, J. et al. *XML Advanced Electronic Signatures (XAdES)*. World Wide Web Consortium (W3C) Note, February 2003. http://www.w3.org/TR/XAdES/ [accessed 8/26/15].

[xmldisg-core] Bartel, M. et al. *XML Signature Syntax and Processing (Second Edition)*. World Wide Web Consortium (W3C) Recommendation, June 2008. http://www.w3.org/TR/xmldsig-core/ [accessed 8/26/15].

[XPATH 2.0] World Wide Web Consortium (W3C) XML Path Language (XPath) 2.0 (Second Edition), December 2010. http://www.w3.org/TR/xpath20 [accessed 8/26/15].

Appendix D—Change Log

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

Release 2 – July 20, 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 guidelines 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 guidelines, 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 guidelines, added GEN-18 and GEN-19.
- Added Section 4.1.1 on specifying the version and version scheme in corpus tags. Included adding new guidelines, 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 guidelines, 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).

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.

Release 3 – August 28, 2015 (Third public comment draft)

Functional Additions/Changes/Removals:

- Created a new Figure 1 to illustrate software lifecycle events and their relationship to SWID tags.
- Revised the descriptions of primary tags, now found in Section 2.1.2, and patch tags, now found in Section 2.1.3.
- Revised the description of the `<Link>` element in Section 2.3.4, and added Table 2 to list the predefined values of the `@rel` attribute.
- Revised the description of the `<Payload>` element in Section 2.3.6 to better distinguish it from the `<Evidence>` element.
- Changed the definitions of *authoritative tag* and *non-authoritative tag* in Section 3.2.
- Deleted the old Section 3.3 on implementing `<SoftwareIdentity>` elements and its GEN-2 requirement, and renumbered the subsequent Section 3.x subsections accordingly.
- Split the old GEN-3 requirement in Section 3.3 into two requirements, GEN-2 and GEN-3, and added new supporting text.
- Added a new GEN-6 requirement to Section 3.4, and renumbered all subsequent GEN requirements.

- Added a new Section 4.1.1 on the proper setting of the <SoftwareIdentity> @corpus attribute, which includes a new COR-1 guideline. Renumbered all subsequent Section 4.1.x subsections and COR guidelines accordingly.
- Split the old COR-3 guideline from the old Section 4.1.1 into new COR-4 and COR-5 guidelines in the new Section 4.1.2.
- Added a new Section 4.2.1 on the proper setting of the <SoftwareIdentity> tag type indicator attributes, which includes a new PRI-1 guideline. Renumbered all subsequent Section 4.2.x subsections and PRI guidelines accordingly.
- Split the old PRI-5 guideline from the old Section 4.2.1 into new PRI-6 and PRI-7 guidelines in the new Section 4.2.2.
- Rewrote the old Section 4.2.3 on specifying attributes required to form CPE names into a new Section 4.2.4 on specifying product metadata needed for targeted searches. This included significant changes to the old PRI-11, now PRI-13.
- Added a new Section 4.3.1 on the proper setting of the <SoftwareIdentity> @patch attribute, which includes a new PAT-1 guideline. Renumbered all subsequent Section 4.3.x subsections and PAT guidelines accordingly.
- Modified the language in the new Section 4.3.3 (old Section 4.3.2) to use “modify” instead of “change” for a type of change that patch tags document. This affected PAT-5 and PAT-6.
- Added a new Section 4.4.1 on the proper setting of the <SoftwareIdentity> @supplemental attribute, which includes a new SUP-1 guideline. Renumbered all subsequent Section 4.4.x subsections and SUP guidelines accordingly.
- Rewrote Section 5 on SWID tag usage scenarios to focus on the use of SWID tags to address key cybersecurity objectives. Highlights of the changes include the following:
    - Revised the introduction to Section 5.
    - Added an introduction for Section 5.1.
    - Rewrote all seven usage scenarios.
    - Introduced a new Section 5.2 on enforcing organizational software policies.
    - Provided a new Section 5.4 that describes how usage scenarios are supported by guidelines from other sections.
    - Added a new Table 3 that illustrates how each guideline from Sections 3 and 4 supports the usage scenarios in Section 5.
- Completely rewrote and greatly expanded Appendix A on forming CPE names.

Editorial Changes:

- Made minor editorial revisions throughout the report.
- Revised the meanings of the terms “guidance” and “guidelines.” *Guidance* now refers to the overall subject matter of the report, and *guidelines* refer to specific items of guidance.
- Switched the order of the old Section 2.1 on SWID tag placement with Section 2.2 on SWID tag types and the software lifecycle.
- Integrated the old Section 2.3 on SWID tag deployment into the new Section 2.2 on SWID tag placement. Renumbered all following Section 2.x subsections accordingly.
- Updated the acronym list (Appendix B) and the references (Appendix C) to take into account related changes made elsewhere in the report.