NISTIR 8060 (Final Public 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 (Final Public 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, Maryland

December 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
103 pages (December 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: December 17, 2015 through January 8, 2016

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 /International Electrotechnical Commission 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, Bob Byers, and Alex J. Nelson of the National Institute of Standards and Technology (NIST); Steve Klos of TagVault.org and 1E; Charles Schmidt of The MITRE Corporation; Piotr Godowski of IBM, and Hopeton Smalling of OQI Cares, Inc. for their reviews of and contributions of feedback to this report.

Note to Reviewers

This document represents a final discussion draft of this report. The authors have conducted a number of iterations of this report to further develop the concepts and guidelines contained herein based on public feedback. This is the final iteration of public review before finalizing this initial revision of the report.

For this final draft, reviewers should focus their reviews on the overall report. Detailed review of all the guidelines in Sections 5 and 6 is also requested to ensure that the guidelines appropriately balance the needs of tag providers and consumers.

Trademark Information

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`>”. Attribute values are indicated in quotations, such as the example “`true`” above.

In cases where any valid value may be provided for an XML attribute, this report specifies "`<any>`" as the attribute value.

This report defines a number of new XML attributes that are extensions to the SWID specification. These extension attributes are defined in a new XML namespace http://csrc.nist.gov/ns/swid/2015-extensions/1.0. These new attributes will be assigned the prefix "n8060" mapped to this namespace. In guidelines and examples, extension attributes will be provided in the form "`@n8060:attributeName`". The schema for these extensions can be found in Appendix A.

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.

Many portions of this document include cross-references to other sections. Such references are often indicated by the use of the "section symbol" (§) or to multiple sections (§§).

Table of Contents

1 Introduction 1
1.1 Problem Statement 1
1.2 SWID Tag Benefits 2
1.3 Purpose and Audience 4
1.4 Section Summary 5
1.5 Report Structure 6
2 SWID Tag Concepts 7
2.1 SWID Tag Types and the Software Lifecycle 7
2.1.1 Corpus Tags 8
2.1.2 Primary Tags 9
2.1.3 Patch Tags 10
2.1.4 Supplemental Tags 12
2.2 SWID Tag Creation 13
2.3 SWID Tag Placement 13
2.3.1 Placement During Installation 13
2.3.2 SWID Tag Generation from Existing Package Management Data 15
2.3.3 Placement in a Repository of SWID Tags 16
2.4 Summary 16
3 SWID Tag Overview 18
3.1 SWID Tag Data Elements 18
3.1.1 <SoftwareIdentity>: The Root of a SWID Tag 18
3.1.2 <SoftwareIdentity> Sub-Element: <Entity> 21
3.1.3 <SoftwareIdentity> Sub-Element: <Evidence> 23
3.1.4 <SoftwareIdentity> Sub-Element: <Link> 24
3.1.5 <SoftwareIdentity> Sub-Element: <Meta> 26
3.1.6 <SoftwareIdentity> Sub-Element: <Payload> 27
3.2 Authenticating SWID Tags 28
3.3 A Complete Primary Tag Example 29
3.4 Summary 30
4 Implementation Guidance for All Tag Creators 32
4.1 Limits on Scope of Guidelines 32

174 4.2 Authoritative and Non-Authoritative Tag Creators 33
175 4.3 Implementing <SoftwareIdentity> Elements 33
176 4.4 Implementing <Entity> Elements 34
177 4.4.1 Providing Detailed Information about Entities 35
178 4.4.2 Preventing Complex Entity Specifications 35
179 4.4.3 Distinguishing Between Authoritative and Non-Authoritative Tags 36
180 4.4.4 Furnishing Information about the Software Creator 37
181 4.5 Implementing <Link> Elements 37
182 4.5.1 Linking a Source Tag to a Known Target Tag 37
183 4.5.2 Linking a Tag to a Collection of Tags 38
184 4.6 Implementing <Payload> and <Evidence> Elements 42
185 4.6.1 Providing Sufficient File Information 42
186 4.6.2 Hash Function Selection 43
187 4.6.3 Handling of Path Separators and Environment Variables 45
188 4.7 Providing Attribute Values in Multiple Languages 46
189 4.7.1 Specifying Product Names in Multiple Languages 47
190 4.7.2 Specifying <Entity> Elements in Multiple Languages 48
191 4.7.3 Specifying <Payload> Elements in Multiple Languages 49
192 4.8 Updating Tags 50
193 4.9 Summary 51
194 **5 Implementation Guidance Specific to Tag Type 52**
195 5.1 Implementing Corpus Tags 52
196 5.1.1 Setting the <SoftwareIdentity> @corpus Attribute 52
197 5.1.2 Specifying the Version and Version Scheme in Corpus Tags 52
198 5.1.3 Specifying the Corpus Tag Payload 54
199 5.2 Implementing Primary Tags 54
200 5.2.1 Setting the <SoftwareIdentity> Tag Type Indicator Attributes 54
201 5.2.2 Specifying the Version and Version Scheme in Primary Tags 54
202 5.2.3 Specifying Primary Tag Payload and Evidence 55
203 5.2.4 Specifying Product Metadata Needed for Targeted Search 57
204 5.3 Implementing Patch Tags 59
205 5.3.1 Setting the <SoftwareIdentity> @patch Attribute 59
206 5.3.2 Linking Patch Tags to Related Tags 59

5.3.3 Specifying Patch Tag Payload and Evidence 59
5.4 Implementing Supplemental Tags 60
5.4.1 Setting the <SoftwareIdentity> @supplemental Attribute 60
5.4.2 Linking Supplemental Tags to Other Tags 61
5.4.3 Establishing Precedence of Information 61
5.5 Summary 62
6 SWID Tag Usage Scenarios 63
6.1 Minimizing Exposure to Publicly-Disclosed Software Vulnerabilities 63
6.1.1 US 1 – Continuously Monitoring Software Inventory 63
6.1.2 US 2 - Ensuring that Products Are Properly Patched 68
6.1.3 US 3 - Correlating Inventory Data with Vulnerability Data to Identify Vulnerable Endpoints 69
6.1.4 US 4 - Discovering Vulnerabilities Due to Orphaned Software Components 71
6.2 Enforcing Organizational Software Policies 72
6.2.1 US 5 - Preventing Installation of Unauthorized or Corrupted Software Products 73
6.2.2 US 6 - Discovering Corrupted Software and Preventing Its Execution 74
6.3 US 7 - Preventing Potentially Vulnerable Endpoints from Connecting to Network Resources 75
6.4 Association of Usage Scenarios with Guidelines 76

List of Appendices

Appendix A— SWID Extension Schema 80
Appendix B— Acronyms 82
Appendix C— References 84
Appendix D— Change Log 86

List of Figures

Figure 1: SWID Tags and the Software Lifecycle 8
Figure 2: Primary Tag Relationships 10
Figure 3: Patch Tag Relationships 11
Figure 4: Supplemental Tag Relations 12

238

239 **List of Tables**

240 Table 1: How Tag Types Are Indicated 20
241 Table 2: <Link> Relations 25
242 Table 3: Allowed Values of @versionScheme 53
243 Table 4: Relationship of Guidelines to Usage Scenarios 78

244

1 Introduction

The International Organization for Standardization (ISO) and the International Electrotechnical Commission (IEC) has published ISO/IEC 19770-2, an international standard for software identification tags, also referred to as SWID tags. A *SWID tag* is a structured set of data elements that identify and describe a software product. The first version of the standard, ISO/IEC 19770-2:2009 [ISO/IEC 19770-2:2009], was published in November 2009. A significantly revised version of the standard, ISO/IEC 19770-2:2015, was published in October 2015, and is referenced herein as the *SWID specification*.

This report provides an overview of the capabilities and usage of SWID tags defined by the ISO/IEC 19770-2:2015 standard. Additionally, this report describes 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 stakeholder benefits that can be gained as SWID tags become more widely produced and consumed within the software 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 and 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 or individual's operational control.

Accurate software inventories of enterprise-managed devices are needed to support higher-level business, information technology, and cybersecurity functions. For example, enterprises need to know how many copies of a given product are installed in order to ensure compliance with software license agreements. To ensure they are not paying for unneeded licenses, enterprises also need to know where specific copies are installed and whether they are in active use. As another example, 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 according to their organizations' security policies. Organizations may also use software inventory information to plan software investments and resources needed to support upgrades to and replacement of legacy systems.

Effective software inventory management depends on the ability to *discover*, *identify*, and *contextualize* software products installed on enterprise-managed devices. Software discovery processes analyze observable states of a managed device to detect and enumerate discrete units of installed software. Discovery is technically challenging due to the enormous variation across the software industry in what it means to be a unit of software. For example, a single unit of

software may consist of a combination of executable files, data files, configuration files, library files, and so forth. A single unit of software may also include supporting software units which may be independently installed and executed, as well as changes to the underlying operating environment, such as the addition of device drivers and entries in operating-system maintained tables and databases. Discovery processes need to be able to handle all these sources of variation while avoiding "false positives" (i.e., erroneous claims that a unit of software is present) as well as "false negatives" (i.e., failures to discover a unit of software that is actually present).

Once the discrete units of software have been enumerated on a device, software identification processes assign identifying labels to those units. These labels are used in various contexts to refer to the products, report their presence, and correlate with other sources of information. A key requirement of the labeling process is that when the same unit of software is discovered on different devices, it must be assigned the same label. Identification is technically challenging because the identifying labels typically are not physically part of the software units and cannot be discovered in the same manner as the software units. Instead, the labels are assigned using inferential techniques based on observable features that vary widely by software provider, operating environment, and device. These inferential techniques may be inaccurate, unreliable, and/or proprietary.

Assuming software units can be discovered and identified, software contextualization processes associate the identifying label with other sources of enriching information. For example, the label of a software unit may be used to collect key descriptive characteristics such as the software unit's exact version, license keys, patch level, associated files in device storage areas, and associated configuration settings. As another example, the assigned software identifier may also be used to search for related patches, upgrades, vulnerabilities and remedies, and configuration checklists. Contextualization is technically challenging to the extent that it depends on widespread agreement on and dissemination of the identifying labels to be assigned to units of software.

The SWID tag standard was developed to help overcome the technical challenges associated with software discovery, identification, and contextualization, and thereby enhance the accuracy and reliability of software asset management processes. SWID tags aid discovery by furnishing a standardized indicator of a software product's presence on a device. Tags aid identification by including a consistent label for a product within its tag. Finally, tags aid contextualization by allowing a wide variety of related product details to be supplied, including the product's full name and version.

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, complete, and normalized data. In other cases, tags may be produced and distributed by other entities, including third parties and through the use of automated tools.

- **Tag consumers:** Organizations and entities that use information contained in SWID tags 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.
- Use stable software identifiers to report changes to a device's software load as changes occur when software is installed, patched, upgraded, and removed.
- Exchange software information between software producers and consumers in a standardized format regardless of software creator, platform, or management tool.
- 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.
- Correlate information about installed software with other information including list(s) of authorized software, related patches, configuration settings, security policies, and advisories.
- Automatically track and manage software license compliance and usage by combining information within a SWID tag with independently-collected software entitlement data.
- Aggregate software asset information for deployed software across an enterprise, providing the organization with knowledge of what software is deployed on specific devices.
- 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, accounts).
- Identify all organizational entities associated with the installation, licensing, maintenance, and management of a software product on an ongoing basis. This identification includes entities external to the software consumer (e.g., software creators, software licensors, packagers, and distributors) as well as those internal to the software consumer's organization.
- Through the optional use of digital signatures, validate that information within a 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 producers can have confidence they are providing all the necessary data, with the requisite data quality, to support the operational goals of each tag usage scenario. Additionally, tag consumers can have confidence that the tags they are using adequately support each tag usage scenario.

This report addresses four 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, to include software developed 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 software build, packaging, and installation tools*, the individuals and organizations that develop tools used by *software providers* to build, package, release, and support installation of software. These tools provide much of the information that is needed to create SWID tags. If these tools generate SWID tags as part of their normal functions, this saves *software providers* from needing to take any specialized actions to produce SWID tags. This makes SWID tag production automatic for any software release managed by the tool, increasing the availability of SWID tags for related products. This is critical for ensuring that SWID tags are provided as part of all commercial and open source software releases. Furthermore, installation tools can support consistent management of installed tags on devices during software installation, upgrade, patch, and removal processes. This report offers guidance on the information that needs to be included in a SWID tag to ensure that tags generated by these tools support SWID tag usage scenarios required by *software consumers*.

The third audience is *providers of inventory-based products and services*, the individuals and organizations that develop tools for discovering, monitoring, and managing software assets for any reason, including securing enterprise networks using standardized inventory information. This audience needs consistency in the content and interpretation of data in SWID tags collected from computing devices to make full use of this information. This report offers guidance to *software providers* on how to consistently implement tags to support SWID tag usage scenarios. The degree of consistency supported by this guidance helps *inventory-based product providers* to use tag data to materially enhance the quality and coverage of software information collected and utilized by their products.

The fourth audience is *software consumers*, the individuals and organizations that install and use commercial, open source, and/or in-house developed software products, and inventory-based products and services. 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 their anticipated usage scenarios. The guidance

provided in this report to *software providers* also ensures that provided tags are useful in meeting the usage scenarios required by *software consumers*.

Through the definition of a set of usage scenarios, this report identifies how the goals of these four audiences are interrelated. Consumers are 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 consumer demand for tag support. Software build and installation tool providers can assist software providers with producing tags for software releases and to manage tags as part of software installation processes. Inventory-based tool implementers are uniquely positioned to recognize how tags can make their products more reliable and effective, and to 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:

- The ISO/IEC 19770-2:2015 international standard specifies the data format for SWID tags described by this report.
- SWID tags were developed to help enterprises meet the need for accurate and complete software inventories to support higher-level business and cybersecurity functions.
- SWID tags provide benefits to organizations that create and use tags.
- Four audiences have interrelated goals pertaining 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.
    - *Providers of software build, packaging, and installation tools* may want to support SWID tags in their tools. This support can assist software providers that use their tools with generating tags during product build and release processes. Additionally, these tools can support the management of installed tags on devices during software installation, upgrade, patch, and removal processes. They need clear guidance on what information needs to be included in a SWID tag to ensure that they generate useful tags for software consumers.
    - *Providers of inventory-based products and services* may want to use SWID tags as their primary method for identifying software. They need more tags to become available to make their specialized tools more reliable and effective based on tag

446 data. They act as software providers as well as software consumers, and thus have
447 the needs and goals of both audiences.

448 - *Software consumers* are trying to cope with the challenges of maintaining an
449 accurate software inventory and addressing software-related management and
450 cybersecurity issues. They need software providers to supply tags along with their
451 products as a common practice.

452 • This report seeks to raise awareness of the SWID tag standard, promote understanding of
453 the business and cybersecurity benefits that may be obtained through increased adoption
454 of software tagging standards and practices, and provide detailed guidance to both
455 producers and consumers of SWID tags to promote consistency and interoperability.

456 ## 1.5 Report Structure

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

458 • Section 2 reviews key SWID tag concepts that are helpful for understanding the different
459 types of tags, how tags are created, and how tags are made available for use. This section
460 will be of interest to all audiences.

461 • Section 3 presents a high-level overview of the SWID tag standard. This section will be
462 of interest to all audiences, as it explains what a SWID tag is and how a tag encodes a
463 variety of identifying and descriptive data elements about a software product.

464 • Section 4 provides implementation guidelines that address common issues related to tag
465 deployment and processing on information systems. These guidelines are intended to be
466 broadly applicable to common IT usage scenarios that are relevant to both public and
467 private sector organizations.

468 • Section 5 provides implementation guidelines for specific types of tags.

469 • Section 6 presents usage scenarios for software asset management and software integrity
470 management. These are not intended to represent an exhaustive or conclusive list of
471 possible SWID applications; they provide informative examples regarding the use of
472 SWID tags based on the SWID specification and guidance in this report to address
473 various organizational needs.

474 • Appendix A provides the XML schema for the extensions defined in this report.

475 • Appendix B presents a list of acronyms used in this report.

476 • Appendix C provides the references for the report.

477 • Appendix D provides the change log for the report.

478

2 SWID Tag Concepts

A SWID tag is a standardized XML format for a set of data elements that identify and describe a software product. 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 of the related versioned software product on that device. The SWID specification defines these behaviors and related behaviors associated with software licensing, patching, and upgrading. This report uses the term *tagged software product* (or, simply, *tagged product*) to refer to a software product that is installed on a device along with one or more discoverable tags describing that product.

This section is intended to provide a general understanding of basic SWID tag concepts. 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 common methods for creating tags. Section 2.3 discusses expectations regarding where SWID tags can be placed after they are created. Finally, Section 2.4 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, different states of software products. These three tag types come into play at different points in the software lifecycle, and 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, the relationship among those lifecycle events, supported by the four types of SWID tags, as follows:

- **Software Deployment.** Before the software product is installed (or *pre-installation*), and while the product is being deployed, a corpus tag provides information about the installation files and distribution media (e.g., CD/DVD, distribution package).
- **Software Installation.** A primary tag will be installed with the software product (or subsequently created) to uniquely identify and describe the software product. Supplemental tags are created to augment primary tags with additional site-specific or extended information. Patch tags provide information about software fixes included with the installation.

[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.

- **Software Patching.** When a new patch is applied to the software product, a new patch tag is provided, supplying details about the patch and any of its dependencies.
- **Software Upgrade.** As a software product is upgraded to a new version, new primary and supplemental tags replace previously deployed tags, enabling timely and accurate tracking of updates to software inventory.
- **Software Removal.** Upon removal of the software product, relevant SWID tags are removed. This removal event can trigger timely updates to software inventory reflecting the product's removal.

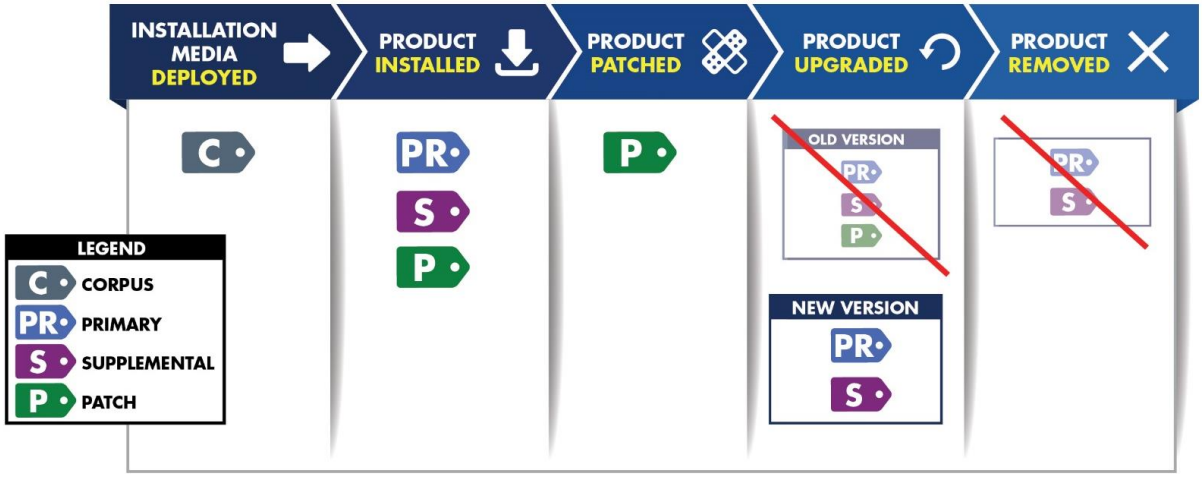

Figure 1: SWID Tags and the Software Lifecycle

The software lifecycle events described in Figure 1 and the tag types related to these events 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, a networked computing device, 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 (see §3.2). If a manifest of the installation files is included in the corpus tag (see §3.1.6 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. All of this information can be used as part of an

automated policy decision to allow or prevent the software installation (see §6.2.1) on an endpoint.

2.1.2 Primary Tags

As illustrated earlier in Figure 1, primary tags are involved in different software lifecycle events. The SWID specification defines *primary tags* to identify and describe software products once they have been successfully installed on an endpoint. The primary tag for each tagged product needs to furnish 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.

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* for the software installation. 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 installed 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.

When a product is upgraded, the primary tag(s) associated with the old version are removed and replaced with a primary tag(s) for the new version. When a product is removed from a device, its primary tag(s) are removed as well. By strictly maintaining the one-to-one association between installed software and associated tags, it is possible to continuously monitor installed software inventory and track software updates using SWID tag data (see §6.1.1).

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 §3.1.4), 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, below.

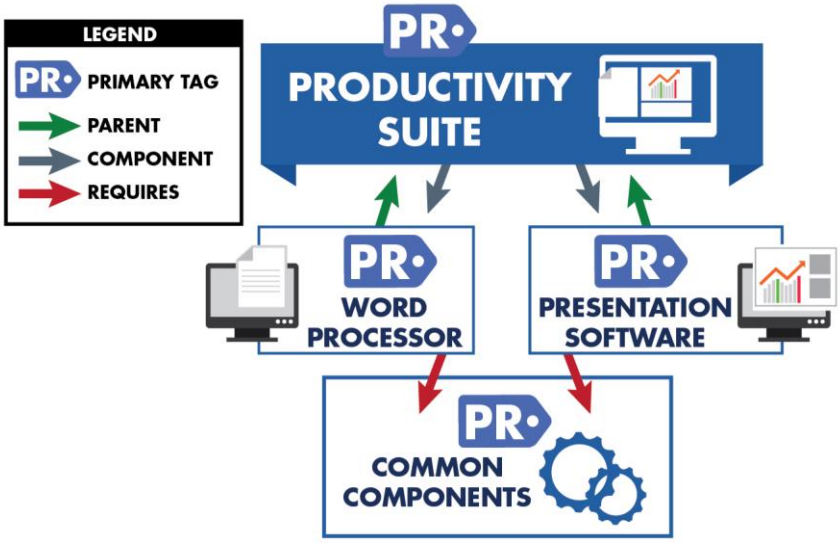

Figure 2: Primary Tag Relationships

This illustration shows how primary SWID tags indicate the associations among several software components. An overarching Productivity Suite will have a **component** relationship with each of its supporting applications, such as a Word Processor and Presentation Software. These subsidiary software products will each have a **parent** relationship to the Productivity Suite. These subsidiary products may also **require** other component software (e.g., language pack, spell checking software).

2.1.3 Patch Tags

A software provider may release patches to correct errors in or add new features to a product. When a patch is installed on a device, changes are made to a product's installation footprint. Since a patch augments an existing installation, these changes need to be tracked separately. The SWID specification defines *patch tags* for software providers to identify and describe each patch. When a patch is installed, a patch tag is placed on the device in the same location where the patched product resides A patch tag can also be created by discovery tools, when a patch tag was not provided by a software provider, to indicate the previous application of a patch. In contrast with a patch, an *upgrade* is defined as a complete replacement of a product's installation footprint. An upgrade typically changes the product's version number and/or release details.

The data elements contained in a patch tag identify and describe the patch rather than the product to which the patch is applied. For example, the product name and version recorded in a patch tag need not match the product name and version recorded in the patched product's primary tag. Instead, these attributes can be used to record the name and version of the patch as assigned by the software provider.

A patch tag can document relationships between the patch described by the patch tag and other products or patches that may be available using the `<Link>` element (see §3.1.4). Three types of relationships are worth noting here:

- **Patches.** Patch tags are required to document a relationship to the primary tag of the patched product that indicates that the patch applies to the patched product. In this way patch tags may assist in determining whether an installed product has all required patches applied. Expressed using a `<Link>` element where the `@rel` attribute is set to "`patches`" with a pointer to the patched product's primary tag.
- **Requires.** To document that a patch described by the patch tag requires the prior installation of another patch. Expressed using a `<Link>` element where the `@rel` attribute is set to "`requires`" with a pointer to the patch tag of the required patch.
- **Supersedes.** To document that a patch described by a patch tag can entirely replace another patch. If the other patch is installed, it will be removed when the new patch is installed. Expressed using a `<Link>` element where the `@rel` attribute is set to `supersedes` with a pointer to the patch tag of the superseded patch using.

When used in this way, patch tags may assist in determining whether an installed product has all required patches applied (see §6.1.2). Figure 3 illustrates the tag relationships for four product patches that can be applied over time.

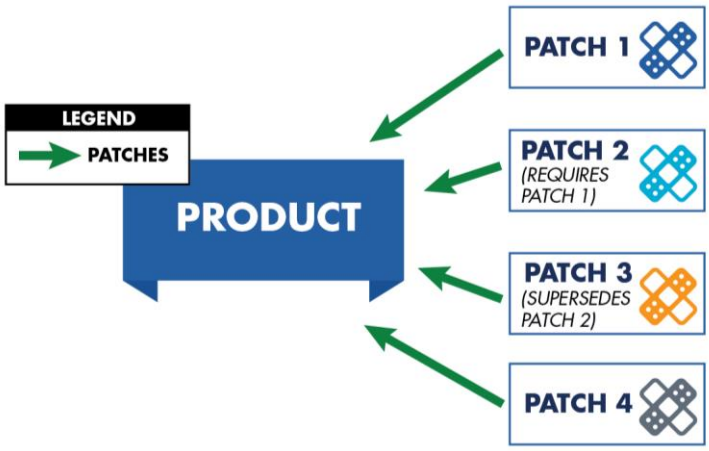

Figure 3: Patch Tag Relationships

All patches have `@rel=patches` Product, since they all patch the same product.

Patch 2 has `@rel=requires` Patch 1, since Patch 1 must be installed before Patch 2.

Patch 3 has `@rel=supersedes` Patch 2, since Patch 3 entirely replaces 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.

A patch will likely also include a manifest of the new and/or changed files (see §3.1.6 for discussion 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 files that misrepresent the installation status of a patch.

2.1.4 Supplemental Tags

The SWID specification requires that tags may not be modified by any entity other than the tag creator. In order to provide a mechanism whereby consumers and software management tools 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.

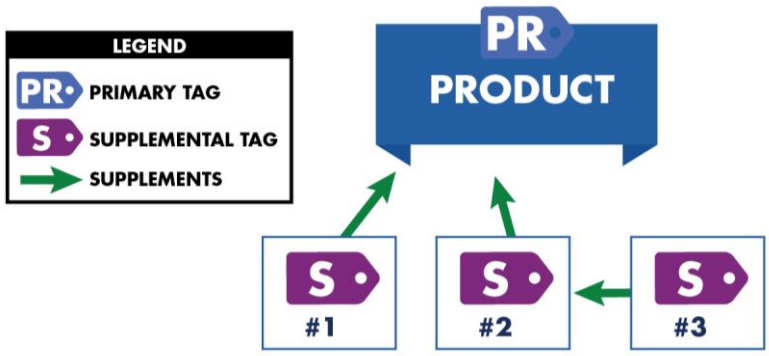

Figure 4: Supplemental Tag Relations

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 (as illustrated in #3 above). 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 ignore conflicting data or to report all conflicting data as exceptions. For example, the mandatory product name recorded in a supplemental tag should

match the product name recorded in the product's primary tag, but if they are different, the name recorded in the primary tag should be used as the most reliable name.

As Figure 1 earlier illustrates, after a software product is upgraded, all primary and patch 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, and the no longer relevant supplemental tags should be removed. When a software product is removed, all primary, patch, and supplemental tags associated with the product should be removed.

2.2 SWID Tag Creation

A SWID tag for a software package, product, or patch 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)
- As the result of a technical analytic process performed by an entity that obtains a copy of a product after its release to market

2.3 SWID Tag Placement

This section describes various factors regarding the placement of SWID tags relative to the software products they describe. Section 2.3.1 describes how and where SWID tags should be placed as the result of installing new software, applying a patch or performing an update to existing software. Section 2.3.2 describes the placement of SWID tags that are generated from existing package management data. Section 2.3.3 provides information about storing SWID tags within a repository separate from a software installation.

2.3.1 Placement During Installation

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. Such a procedure may be run as the result of installing new software, or by applying a patch or update to existing software. This method of tag deployment is available when the tag creator is in a position to ensure that the tag is included in the installation package.

During software installation a SWID tag is placed relative to the product it identifies and describes. The SWID specification makes the following statements about SWID tag placement in this situation:

> 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. 6-7].

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 SWID specification provides the following rules and recommendations that shall be used when creating names for SWID tag files:

> Filenames should be restricted to use only the characters listed in the Portable Filename Character Set defined in IEEE 1003.1:2013, 3.278 to maximize interoperability between platforms. If this limitation is too restrictive, the tag creator shall ensure that the characters used in the filename are valid characters for all platforms where their SWID tags may be stored on a file system. SWID tag base filenames (i.e. the filename without the .swidtag extension) shall be structured to be globally unique for the tag creator and product. SWID tag creators may use different approaches to defining the base portion of the SWID tag filename; however, if the filename aligns with the following structure, the filename will be unique for the product and recognizable by a system administrator <name of the tag creator> + <product name>.swidtag. The .swidtag file extension shall be used for all software identification tags. [ISO/IEC 19770-2:2015, p. 8]

Following this guidance, for example, the SWID tag referencing product name "ACME Roadrunner" from the software creator "acme.com" would be stored in the filename, "`acme.com.acmeroadrunner.swidtag`".

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

[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.

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.

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 the same as a powered-down physical machine, effectively rendering the installed products unavailable, but not removing them from the software inventory. In contrast, destroying a virtual machine is treated the same as the decommissioning of a physical machine. In the latter case, all installed software products are removed from the software inventory.

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 placed during installation processes is that tags reside on the same physical or virtual storage device as where the tagged product resides. Although tag consumers may infer that a product is executable on the same device where it is stored, they will benefit from distinguishing cases where products may be executable on devices elsewhere within the enterprise.

2.3.2 SWID Tag Generation from Existing Package Management Data

A second 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 device, as long as the native package manager provides a published interface allowing valid tags to be obtained. When a tag is produced dynamically on the installation host in this way, it will not be possible for this tag to be digitally signed. As a result, it will not be possible to verify the integrity of the tag based on a digital signature unless an equivalent tag is also produced using the first method described above. Without being able to verify the tag's signature, processes on the device and downstream consumers that use the information within a SWID tag will have less assurance of the tags authenticity and integrity. This will limit the usefulness of the SWID tag information to address some usage scenarios that require a high degree of assurance.

2.3.3 Placement in a Repository of SWID Tags

A third method of tag deployment is to store SWID tags in accessible repositories. By retrieving specific tags from an appropriate repository, software consumers can do the following:

- Confirm that a tag which has been discovered on an endpoint has not been modified; this can done by comparing the tag found on the endpoint with the same tag found in the repository
- 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

2.4 Summary

This section covered basic SWID tag concepts. The SWID specification defines four different types of SWID tags to support various portions of the software lifecycle, including pre-installation, product installation, patching, software updates, and software decommissioning.

- Corpus SWID tags provide information about a software distribution, to include information about the files that are included as part of the distribution. This information may be used to verify the integrity and/or authenticity of the software distribution.
- Primary SWID tags provide specific information (e.g., product naming information, the software creator, lists of files expected to be included) regarding a software product that has been installed.
- Patch SWID tags provide specific information about a software patch that has been provided to correct errors in or add new features to a product. A patch tag supplies information about the changes that a patch makes to the patched software product's installation footprint including files added, removed, or changed.
- Supplemental SWID tags provide a mechanism to describe additional information related to tags of other types.

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.

By placing SWID tags in consistent locations relative to the software products they identify, inventory processes and automated tools are able to consistently and accurately maintain software inventory.

SWID tags may also be placed in accessible repositories to make tag information available to be used by other management and cybersecurity processes.

3 SWID Tag Overview

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). When used with a validating XML parser, the XSD can be used to check that a SWID tag is conformant with the SWID specification. The XSD may be downloaded from:

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

This section is intended to provide an overview of the data model used to express and authenticate SWID tags.

The remainder of this section is organized as follows. Section 3.1 presents an overview of the basic data elements that comprise a SWID tag. Section 3.2 discusses how SWID tags may be authenticated. Section 3.3 presents an example of the primary tag type, and Section 3.4 concludes with a summary of key points from this section.

3.1 SWID Tag Data 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 3.1.1. The following sub-elements are used to express distinct categories of product information: `<Entity>` (see §3.1.2), `<Evidence>` (see §3.1.3), `<Link>` (see §3.1.4), `<Meta>` (see §3.1.5), and `<Payload>` (see §3.1.6).

3.1.1 <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`”. Sections 5.1.1 and 5.2.1 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. Typical reasons for an increment in `@tagVersion` include: 1) fixes of erroneous information in an older version of a tag; or, 2) the addition of new information that was not included in a previously deployed tag. A value for `@tagVersion` is **optional** and defaults to "`0`".

  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, by having the same `@tagId` attribute value, but different `@tagVersion` attribute 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 most up-to-date tag, and the others may be ignored. When considering tags in this situation, it is important to verify tag signatures, if available, to ensure that the most up-to-date tag being considered contains a valid XML signature (see §3.2). Furthermore, it is important that this signature contain a valid certificate to avoid using a tag that might have been produced by an unauthorized party. For example, if the most-up-to-date tag is found to contain an invalid signature, then a valid tag with the next lowest `@tagVersion` value is likely to be safer to use.

- `@supplemental`: a boolean value that, if set to "`true`", indicates that the tag type is *supplemental* (see §2.1.4). 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* (see §2.1.3). 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* (see §2.1.1). 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 §§ 5.1.1, 5.2.1, 5.3.1, and 5.4.1). In Sections 5.3.1 and 5.4.1 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

3.1.1.1 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 (see §3.1.2) is included so the example illustrates all data values required in a minimal tag that conforms to the SWID specification. 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>
```

3.1.1.2 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 (see §3.1.2) is included so the example illustrates all data values required in a minimal tag that conforms to the standard. The `<Link>` element (see §3.1.4) is included to illustrate how a supplemental tag may be associated with the primary tag shown above in Section 3.1.1.1. 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>
```

3.1.1.3 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>
```

3.1.2 <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` can be selected from an extensible set of allowed tokens, including these:
    - **`aggregator:`** An organization or system that encapsulates software from their own and/or other organizations into a different distribution process (as in the case of virtualization), or as a completed bundle to accomplish a specific task (as in the case of a value added reseller)
    - **`distributor:`** An entity that furthers the marketing, selling and/or distribution of software from the original place of manufacture to the ultimate user without modifying the software, its packaging or its labelling
    - **`licensor:`** A person or organization that owns or holds the rights to issue a software license for a specific software package
    - **`softwareCreator:`** A person or organization that creates a software product
    - **`tagCreator:`** The entity that creates a given SWID tag
- `@thumbprint`: for SWID tags which are digitally signed (see §4.6), this value provides a hexadecimal string containing a hash of the entity's signing certificate. This allows the digital signature to be directly related to the entity specified.

Values for `@regid` are URIs as described in RFC 3986 [RFC 3986]. Values for `@regid` are not required to be dereferenceable on the Internet. 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>
```

3.1.3 <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> (see §3.1.6). 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 location=".." name="rrdetector">
      <File name="rrdetector.exe" size="532712"/>
      <File name="sensors.dll" size="13295"/>
    </Directory>
  </Evidence>
  …
</SoftwareIdentity>
```

In cases where the evidence is collected from a shared location (e.g., a NAS device), the provided @deviceId could reference that shared location rather than the endpoint where the discovery occurs. Using this approach helps to prevent a situation where software installed in a shared location is tagged multiple times with conflicting @deviceId values.

3.1.4 <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 §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 a number of attributes, two of which are required, as follows:

- @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]

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

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

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

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

3.1.5 <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>
```

3.1.6 <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 (see §3.1.3). When the <Payload> element is used, 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%" name="rrdetector">
        <File name="EPV12.cab" size="1024000"
              SHA256:hash="a314fc2dc663ae7a6b6bc6787594057396e
6b3f569cd50fd5ddb4d1bbafd2b6a" />
```

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

    </Directory>
  </Payload>
  …
</SoftwareIdentity>
```

3.2 Authenticating SWID Tags

Because SWID tags are XML documents discoverable on a device (see §2.3.1, §2.3.2) or by retrieving a tag from a repository (see §2.3.3), 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 was produced by a known, trusted entity and has not been altered after creation. XML digital signatures embedded within a SWID tag can be used to prove the authenticity of the tag signer and to validate that changes have not been made to the original tag that was signed.

Applying an XML digital signature to a tag directly after it is created addresses the following risks:

- **Creation of a tag by an unauthorized creator.** Any entity can create a tag for a software product. By including a certificate acquired from a trusted certificate authority in the tag's digital signature, it is possible to validate the certificate, ensure that it has not been revoked, and determine that the signature was produced while the certificate is valid. By linking the signature to the `<Entity>` element having a `@role` value of "`tagCreator`" and using the `@thumbprint` attribute (see §3.1.2) to record the thumbprint of the certificate, it is possible to identify that the tag creator is also the tag signer. This provides a method for source authentication of a tag's creator.
- **Detecting unauthorized changes to a tag**. Changes made to a tag after it is created can be detected by validating the tag's signature. Such changes may occur due to accidental modification, incomplete copying, network errors, filesystem corruption, or by an attacker wishing to misrepresent the installation state of installed software or the information contained within a tag. The use of an XML digital signature in this way provides a means to measure the integrity of a tag. When coupled with authenticating the creator of a tag, this greatly increases the assurance of the tag data.
- **Detecting unauthorized changes to software installation media and packages.** If the creator of a corpus tag can be authenticated and the integrity of the tag can be verified, it is then possible to use the information in the tag's `<Payload>` element, if provided, to measure the integrity of software installation media or packages. This can be useful for detecting unauthorized changes to software installation media and packages, and can inform policy decisions pertaining to authorizing software installations (see §6.2.1).
- **Detecting unauthorized changes to installed software**. If the creators of primary and patch tags related to an installed software product can be authenticated and the integrity of the tags can be verified, it is then possible to use the information in each tag's `<Payload>` element, if provided, to measure the integrity of the related installed

> software product. This can be useful for detecting unauthorized changes to installed software and can inform policy decisions pertaining to authorizing software execution (see §6.2.2).

When considering the totality of these risks, SWID tags can enhance the assurance of software before and after it is installed, in a standardized way, regardless of the target platform or software product installed.

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.

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 also requires that a digitally-signed SWID tag enable tag consumers to:

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

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

3.3 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.

---

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

```
<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%" name="rrdetector">
      <File name="rrdetector.exe" size="532712"
        SHA256:hash="a314fc2dc663ae7a6b6bc6787594057396e6b3f569c
d50fd5ddb4d1bbafd2b6a"/>
      <File name="sensors.dll" size="13295"
        SHA256:hash="54e6c3f569cd50fd5ddb4d1bbafd2b6ac4128c2dc66
3ae7a6b6bc67875940573"/>
    </Directory>
  </Payload>
</SoftwareIdentity>
```

3.4 Summary

SWID tags are rich sources of information useful for identifying and describing software products, whether in a pre-installation state or when 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 SWID 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

1292 <Evidence> for use by scanning tools to record results of the discovery process, <Link> to
1293 associate tags with other items, including other tags, <Meta> to record a variety of metadata
1294 values, and <Payload> to enumerate files, etc., that comprise the installed product. Finally,
1295 digital signatures may optionally be used by any tag producer to ensure that the contents of a tag
1296 are not accidentally or deliberately modified after installation, and to provide authentication of
1297 the signer.

4 Implementation Guidance for All Tag Creators

Sections 4 and 5 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 address common cybersecurity and operational IT usage scenarios, such as those defined in Section 6. In doing so this will satisfy the tag handling requirements of both public and private sector organizations.

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 §5.1)
- **PRI:** Guidelines specific to primary tags (see §5.2)
- **PAT:** Guidelines specific to patch tags (see §5.3)
- **SUP:** Guidelines specific to supplemental tags (see §5.4)

This section provides implementation guidelines that address issues common to all situations in which tags are deployed and processed. Section 5 provides guidelines that vary according to the type of tag being implemented.

4.1 Limits on Scope of Guidelines

This report assumes that tag creators 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.

4.2 Authoritative and Non-Authoritative Tag Creators

SWID tags may be created by individuals, organizations, or automated tools under different conditions. The entity that creates a tag, as well as the conditions under which a tag is created, profoundly affect how it can be used based on 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 a first- or second-party that creates a tag as part of the process of releasing software. A first-party authoritative tag creator is the software creator. A second party authoritative tag creator aggregates, distributes, or licenses software on behalf of the software creator. Such parties typically possess accurate, complete, and detailed technical knowledge that is needed for creation of authoritative tags containing reliable information.

A *non-authoritative tag creator* is defined as an entity 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 forensic methods while discovering installed software.

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 equally 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.

4.3 Implementing <SoftwareIdentity> Elements

This section provides guidelines to be observed by tag creators when implementing SWID tag `<SoftwareIdentity>` elements.

The SWID specification defines an international standard intended to be adopted and used worldwide. To support an international audience it is necessary to permit tag creators to provide language-dependent attribute values in region-specific human languages. For example, a Japanese software provider may want to specify the value of a particular product’s `<SoftwareIdentity>` `@name` attribute as a string of Japanese characters.

The SWID tag XML schema provides multi-language support in two ways. First, the schema specifies UTF-8 as the allowed character encoding scheme for SWID tag files. Second, the schema allows the optional `@xml:lang` attribute to be included on all tag elements. By taking advantage of these features, a Japanese software provider could issue a SWID tag like this:

```
<SoftwareIdentity
  xmlns="http://standards.iso.org/iso/19770/-2/2015/schema.xsd"
  xml:lang="ja-jp"
  name="コンピュータ管理システム 2015"
  tagId="jp.largecomputerco.タグ番号 1"
  version="1.0">
```

```
  <Entity
    name="大型コンピュータ会社"
    regid="largecomputerco.jp"
    role="tagCreator softwareCreator"/>
  …
</SoftwareIdentity>
```

According to W3C documentation, *language tags* "are used to indicate the language of text or other items in HTML and XML documents" [W3C-langtags]. By supplying the language tag "`ja-jp`" as the value of the `<SoftwareIdentity>` `@xml:lang` attribute, the tag creator signals to tag consumers that various language-dependent attributes, such as the `<SoftwareIdentity>` `@name` attribute, are provided in Japanese. Additionally, the SWID schema is designed such that any value of `@xml:lang` specified on any tag element is inherited by all of that element's child elements, unless explicitly overridden. As a result, the value specified for the `<SoftwareIdentity>` `@xml:lang` attribute will, in effect, establish the *preferred language* that is used for all language-dependent attributes within the tag.

Knowing the preferred language of SWID tag attribute values can be very useful to tag consumers, and can relieve them of the need to attempt to perform language auto-detection. The following guideline requires that tag creators always specify the preferred language of a tag:

> **GEN-2.** Every `<SoftwareIdentity>` element MUST specify a `@xml:lang` attribute with a non-blank value to indicate the default human language used for expressing all language-dependent attribute values.

Guideline **GEN-2** is directed towards both authoritative and non-authoritative tag creators. While authoritative tag creators can always be expected to know the default language of the tag, non-authoritative creators may need to use local knowledge to ascertain the most appropriate default language. This document recommends that non-authoritative tag creators supply the default language of the device where the tagged product resides.

> **GEN-3.** [Non-Auth] When specifying a value for the `<SoftwareIdentity>` `@xml:lang` attribute, non-authoritative tag creators SHOULD use the language tag corresponding to the default language of the device where the tagged product resides.

In some cases, tag creators may want to specify various SWID tag attribute values in more than one human language. This presents a number of challenges and potential interoperability issues; these are discussed further in Section 4.7.

4.4 Implementing <Entity> Elements

This section provides guidelines to be observed by tag creators when implementing SWID tag `<Entity>` elements. The guidelines in this section address four issues:

1. Providing detailed information about entities (see §4.4.1)
2. Preventing unnecessarily complex entity specifications (see §4.4.2)
3. Distinguishing between authoritative and non-authoritative tags (see §4.4.3)
4. Furnishing information about the software creator (see §4.4.4)

4.4.1 Providing Detailed Information about Entities

The first issue to be addressed concerns the need for detailed information about the various entities associated with tags and/or tagged products.

Section 8.2 of the SWID specification requires that every SWID tag contain an `<Entity>` element where the `@role` attribute has the value "`tagCreator`", and the `@name` attribute is also provided. Although `<Entity>` elements may furnish `@regid` attribute values, the specification does not require this information to be provided. Instead, the specification defaults the `@regid` attribute value to "`http://invalid.unavailable`" when a value is not explicitly provided in the element. Because the `@regid` attribute serves as a unique reference to an organization, this report provides guidelines to ensure that explicit values are provided whenever possible.

The ability to provide `@regid` attribute values varies depending on whether a tag creator is authoritative or non-authoritative. For authoritative tag creators, it is reasonable to require the `@regid` value on all `<Entity>` elements:

> **GEN-4.** [Auth] Every `<Entity>` element MUST provide an explicit (i.e., non-default) `@regid` attribute value.

Non-authoritative tag creators may be less able to provide reliable `@regid` information. While they can be expected to provide a `@regid` value for the `<Entity>` element with the "`tagCreator`" `@role` that identifies their organization, they can only be encouraged to provide `@regid` information for entities in other roles. This leads to the following two guidelines:

> **GEN-5.** [Non-Auth] Every `<Entity>` element SHOULD provide an explicit (i.e., non-default) `@regid` attribute value when such a value can be determined.

> **GEN-6.** [Non-Auth] The `<Entity>` element containing the `@role` "`tagCreator`" MUST provide an explicit (i.e., non-default) `@regid` attribute value.

4.4.2 Preventing Complex Entity Specifications

The second issue to be addressed concerns the potential for unnecessarily complex specifications of entities and roles associated with the tag and/or the tagged product. The SWID specification allows a tag to furnish multiple `<Entity>` elements in order to support situations in which different organizations play different roles with respect to the tag and/or the tagged product. So if one organization were the tag creator and a second organization were the software creator, this information could be specified as follows:

```
<Entity
  name="Organization 1 Corp"
  role="tagCreator"/>
<Entity
  name="Organization 2 Corp"
  role="softwareCreator"/>
```

This ability to furnish multiple <Entity> elements has an undesirable side effect, however. It now becomes possible for role information associated with a single organization to be spread across multiple <Entity> elements, as illustrated by this example:

```
<Entity
  name="Organization 1 Corp"
  role="tagCreator"/>
<Entity
  name="Organization 1 Corp"
  role="softwareCreator"/>
```

Such spreading of role information across <Entity> elements is discouraged. Furnishing entity information in this way increases the size of the tag unnecessarily as well as creates additional processing complexity for tag consumers. To preclude this, the following guideline is provided:

> **GEN-7.** All <Entity> elements that provide the same @regid attribute value MUST provide the same @role attribute values.

Guideline **GEN-7** works in concert with guidelines **GEN-4** through **GEN-6** to achieve the desired effect. It should be clear that the following example satisfies these guidelines:

```
<Entity
  name="Organization 1 Corp"
  regid="org1.com"
  role="tagCreator softwareCreator"/>
<Entity
  name="Organization 2 Corp"
  regid="org2.com"
  role="licensor"/>
```

As will be seen later in Section 4.7, guidelines **GEN-4** through **GEN-7** also play an important role in addressing potential interoperability issues that could arise when tag creators specify attribute values in multiple human languages.

4.4.3 Distinguishing Between Authoritative and Non-Authoritative Tags

The third issue to be addressed here concerns the process by which a tag consumer may rapidly determine whether the tag creator is authoritative or non-authoritative.

When a tag contains an <Entity> element that specifies only a single @role of "tagCreator", tag consumers can safely 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 an <Entity> element that indicates that the entity having the "tagCreator" role also has one or more of these additional predefined roles: "aggregator", "distributor", "licensor", or "softwareCreator".

GEN-8. [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.

4.4.4 Furnishing Information about the Software Creator

The fourth issue to be addressed here concerns the furnishing of information about the software creator, when that information is known.

Explicit knowledge of the software creator is important for many software inventory scenarios, but the SWID specification does not require that this information be provided. To support software inventory scenarios, authoritative tag creators are expected to furnish information on the software creator's identity. If the tag creator is not the same as the software creator, authoritative tag creators are expected to know the appropriate @name and @regid attribute values for the software creator.

GEN-9. [Auth] Authoritative tag creators MUST provide an <Entity> element where the @role attribute contains the value "softwareCreator".

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 knowing a product's software creator, non-authoritative tag creators are encouraged to include this information in a tag whenever possible.

GEN-10. [Non-Auth] Non-authoritative tag creators SHOULD provide an <Entity> element where the @role attribute contains the value "softwareCreator". If known, the @name attribute SHOULD also be provided.

4.5 Implementing <Link> Elements

This section provides guidelines to be observed by tag creators when implementing SWID tag <Link> elements. As discussed in Section 3.1.4, the <Link> element is used to establish relationships of various kinds between tags and other documents, including other tags. The guidelines in this section address two issues:

1. Linking a source tag to a known target tag
2. Linking a tag to a collection of tags

4.5.1 Linking a Source Tag to a Known Target Tag

In many tag creation situations, there will be a need to embed in a source tag (i.e., a tag being created) a <Link> element that points to a preexisting target tag. For example, when a patch tag (see §2.1.3) is being created, it is important to link that patch tag (the source tag) to the tag which

describes the product being patched (the target tag). As another example, a supplemental tag (see §2.1.4) will typically be linked to another preexisting tag whose information is being supplemented. To establish these relationships, tag creators use the `<Link>` `@href` attribute in the source tag to provide a pointer to the target tag.

In these situations, the tag creator will know the contents of the target tag, including its `@tagId`. In this sense, the target tag is known to the tag creator at the time that the source tag is being created. To link a source tag to a known target tag, tag creators are required to use the '`swid:`' scheme followed by the `@tagId` of the target tag.

GEN-11. In order to link a source tag to a specific target tag whose `@tagId` is known at the time the source tag is created, tag creators MUST set the value of the `<Link>` `@href` attribute in the source tag to a URI with "`swid:`" as its scheme, followed by the `@tagId` of the target tag.

This idea is illustrated below with two tag fragments.

Tag 1:

```
<SoftwareIdentity
  name="Application 1"
  tagId="com.largecomputerco.app1"
  …
</SoftwareIdentity>
```

Tag 2:

```
<SoftwareIdentity
  name="Application 2"
  tagId="com.largecomputerco.app2"
  …
  <Link rel="relation" href="swid:com.largecomputerco.app1"/>
</SoftwareIdentity>
```

In the above example, Tag 2 (describing "Application 2") is the source tag, and is linked to the target Tag 1 (describing "Application 1").

4.5.2 Linking a Tag to a Collection of Tags

In contrast to situations where the `@tagId` of the target tag is known, there are also many tag creation situations where there is a need to embed in a source tag a `<Link>` element that points either to a single target tag or to a *collection* of target tags whose `@tagId` value(s) cannot be known with certainty at the time the source tag is created. Consider the following scenarios:

- A primary tag (the source) is being created for a product that requires a commonly used shared library. This shared library is maintained by a third party, has its own primary tag, and is periodically upgraded in ways that maintain backwards compatibility. It is important that the source tag includes a link to the shared library's tag (the target), but it is not possible to specify a fixed `@tagId` for the target tag.

- A patch tag (the source) is being created for a patch that applies to a collection of products based on version. For example, imagine that a software provider has released a series of versions of a product, all in the "4.1.x" version series. A flaw is discovered in version 4.1.5 of the product, and it is determined that the flaw was introduced in version 4.1.0. A single patch is developed to correct this flaw, so its patch tag needs to link to all affected versions.

The SWID specification provides a mechanism that tag creators may use when linking a source tag to a collection of target tags. That mechanism is to set the `<Link>` `@href` attribute value to a URI with a scheme of '`swidpath:`', followed by an XPath 2.0 [XPath20] conformant query. Any such XPath query is expected to be used by a system to iterate over a set of SWID tags and identify matching tags by applying the XPath query to each tag and checking for a non-empty result. Because the SWID specification is not clear and specific about this usage, this document provides the following guidelines:

GEN-12. When linking a source tag to one or more target tags whose `@tagId` value(s) cannot be determined at the time the source tag is created, tag creators MUST set the value of the `<Link>` `@href` attribute in the source tag to a URI with `swidpath:` as its scheme, followed by an XPath 2.0 [XPath20] conformant query. All characters contained in the XPath query which the URI specification [RFC3986] designates as *reserved* MUST be percent encoded per the URI specification. All embedded SWID tag elements in the query MUST be prefixed with the "`swid:`" namespace.

The above guideline clarifies two points: (1) that URI reserved characters in the embedded XPath query must be percent encoded, and (2) that the "`swid:`" namespace must be used for all SWID elements. Thus, in order to process such a query, the "`swidpath:`" scheme must be stripped off, any embedded percent encodings must be replaced with the encoded characters, and the XPath query processor must be supplied with the definition of the "`swid:`" namespace.

The following guideline advises query developers on how to prepare queries in a consistent and interoperable manner.

GEN-13. Any XPath query used within a `<Link>` `@href` element MUST be designed in such a way that it can used by a system to iterate over a set of SWID tags and identify matching tags by applying the XPath query to each tag and checking for a non-empty result.

Next, a series of examples are presented to help the reader to understand the guidelines presented in this subsection.

4.5.2.1 An Example Tag

To illustrate the guidelines for linking tags, a series of examples are presented relative to this example tag:

```
<SoftwareIdentity
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns="http://standards.iso.org/iso/19770/-2/2015/schema.xsd"
  xmlns:SHA256="http://www.w3.org/2001/04/xmlenc#sha256"
```

```
    xmlns:ext="http://example.org/ns/swid-example"
    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"/>
    <Meta
      activationStatus="trial"
      product="Roadrunner Detector"
      colloquialVersion="2013"
      edition="coyote"
      revision="sp1"
      ext:newattr="newvalue"/>
    <Payload>
      <Directory root="%programdata%" name="rrdetector">
        <File name="rrdetector.exe" size="532712"
SHA256:hash="a314fc2dc663ae7a6b6bc6787594057396e6b3f569cd50fd5dd
b4d1bbafd2b6a"/>
      <File name="sensors.dll" size="13295"
SHA256:hash="54e6c3f569cd50fd5ddb4d1bbafd2b6ac4128c2dc663ae7a6b6
bc67875940573"/>
      </Directory>
    </Payload>
</SoftwareIdentity>
```

4.5.2.2 Example 1: Using an XPath Query to Refer to a Tag by its @tagId

Although the ‘`swid:`’ scheme is intended to be used in cases where the target tag’s `@tagId` value is known, the ‘`swidpath:`’ scheme followed by an XPath query could also be used to achieve the same effect. The following XPath query will match the tag illustrated above in Section GEN-12:

```
/swid:SoftwareIdentity[@tagId='com.acme.rrd2013-ce-sp1-v4-1-5-
0']
```

When incorporated into a `<Link>` element, the above query must be prefixed with the ‘`swidpath:`’ scheme, and reserved characters must be percent encoded, as follows:

```
<Link rel="relation"
href="swidpath:%2Fswid%3ASoftwareIdentity%5BtagId%3D%27com.acme.
rrd2013-ce-sp1-v4-1-5-0%27%5D"/>
```

4.5.2.3 Example 2: Using an XPath Query to Refer to a Tag by Name and Tag Creator @regid

In the next example, an XPath 2.0 query is shown which matches any tag such that (1) the product name includes the string "`Roadrunner Detector`", and (2) the `<Entity>` element containing the "`tagCreator`" `@role` also contains the `@regid` value "`acme.com`":

```
/swid:SoftwareIdentity[contains(@name,'Roadrunner Detector')
  and swid:Entity[@regid='acme.com'
  and count(index-of(@role,'tagCreator')) gt 0]]
```

Note: The query above is designed with the assumption that the SWID tag has been validated against the SWID schema before the XPath query engine is run. The final draft of this report will address this topic, and present guidance on whether or not queries should assume tag schema validation.

When incorporated into a `<Link>` element, the above query must be prefixed with the '`swidpath:`' scheme, and reserved characters must be percent encoded, as follows:

```
<Link rel="relation"
href="swidpath%2Fswid%3ASoftwareIdentity%5Bdescendant%3A%3Aswid%
3AFile%5B%40name%20%3D%20%27rrdetector.exe%27%20and%20%40*%5Bloc
al-name()%20%3D%20%27hash%27%20and%20namespace-
uri()%20%3D%20%27http%3A%2F%2Fwww.w3.org%2F2001%2F04%2Fxmlenc%23
sha256%27%20and%20.%3D%27a314fc2dc663ae7a6b6bc6787594057396e6b3f
569cd50fd5ddb4d1bbafd2b6a%27%5D%5D%5D"/>
```

4.5.2.4 Example 3: Using an XPath Query to Refer to a Tag Containing a Known File

In the next example, an XPath 2.0 query is shown which matches any tag that includes a `<File>` element describing the "`rrdetector.exe`" file with a specific hash value:

```
/swid:SoftwareIdentity[descendant::swid:File[@name =
'rrdetector.exe'
  and @*[local-name() = 'hash'
  and namespace-uri() =
'http://www.w3.org/2001/04/xmlenc#sha256'
  and
.='a314fc2dc663ae7a6b6bc6787594057396e6b3f569cd50fd5ddb4d1bbafd2
b6a']]]
```

Such a query could be used in a patch tag that affects a specific file that is used by any number of other products, which may or may not be installed on a device at any given time. When incorporated into a `<Link>` element, the above query must be prefixed with the "`swidpath:`" scheme, and reserved characters must be percent encoded, resulting in this `<Link>` element:

```
<Link rel="relation"
href="%2Fswid%3ASoftwareIdentity%5Bdescendant%3A%3Aswid%3AFile%5
B%40name%20%3D%20%27rrdetector.exe%27%20and%20%40*%5Blocal-
name()%20%3D%20%27hash%27%20and%20namespace-
uri()%20%3D%20%27http%3A%2F%2Fwww.w3.org%2F2001%2F04%2Fxmlenc%23
sha256%27%20and%20.%3D%27a314fc2dc663ae7a6b6bc6787594057396e6b3f
569cd50fd5ddb4d1bbafd2b6a%27%5D%5D%5D"/>
```

4.5.2.5 Example 4: Using an XPath Query to Refer to Tags with a Range of Versions

In the next example, an XPath 2.0 query is shown which matches any tag that describes a product whose name is "Roadrunner Detector" and whose version is 4.1.0 or greater within the 4.x release branch.

```
/swid:SoftwareIdentity[swid:Meta[@product = 'Roadrunner
Detector']
  and tokenize(@version,'\.')[1] cast as xs:unsignedInt = 4
  and tokenize(@version,'\.')[2] cast as xs:unsignedInt ge 1]
```

When incorporated into a <Link> element, the above query must be prefixed with the "swidpath:" scheme, and reserved characters must be percent encoded, resulting in this <Link> element:

```
<Link rel="relation"
href="%2Fswid%3ASoftwareIdentity%5Bswid%3AMeta%5B%40product%20%3
D%20%27Roadrunner%20Detector%27%5D%20and%20tokenize(%40version%2
C%27%5C.%27)%5B1%5D%20cast%20as%20xs%3AunsignedInt%20%3D%204%20a
nd%20tokenize(%40version%2C%27%5C.%27)%5B2%5D%20cast%20as%20xs%3
AunsignedInt%20ge%201%5D%0A"/>
```

4.6 Implementing <Payload> and <Evidence> Elements

This section provides guidelines to be observed by tag creators when implementing SWID tag <Payload> and <Evidence> elements. The guidelines in this section address three issues:

1. Providing sufficient file information (see §4.6.1)
2. Hash function selection (see §4.6.2)
3. Handling of path separators and environment variables (see §4.6.3)

4.6.1 Providing Sufficient File Information

The first issue to be addressed here concerns the amount of payload/evidence information that needs to be provided.

Authoritative tag creators use the <Payload> element to enumerate the files and folders comprising a product or patch, whereas non-authoritative tag creators use the <Evidence> element for this purpose. Files are described using the <File> element, and folders are described using the <Directory> element.

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

GEN-14. 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.

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

GEN-15. Every <File> element provided within a <Payload> or <Evidence> element MUST include a value for the @version attribute, if one exists for the file.

Because improper changes may also occur in ways that do not alter file sizes or versions, file hash values are also necessary. If there is a difference in the files' sizes, a change has occurred. If the size is the same, re-computing a hash will be necessary to determine if a change has occurred. Authoritative tag creators are expected to have sufficient knowledge of product details to be able to routinely provide hash values. Non-authoritative tag creators may not have the necessary knowledge of or access to files to provide hash information, but are encouraged to do so whenever possible.

GEN-16. [Auth] Every <File> element within a <Payload> element MUST include a hash value.

GEN-17. [Non-Auth] Every <File> element within an <Evidence> element SHOULD include a hash value.

4.6.2 Hash Function Selection

The second issue to be addressed here concerns selection of the hash function to be used when providing hash values.

Software products tend to be used long beyond the formal product support period. When selecting a hash function, it is important to consider the deployment 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. 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[5].

GEN-18. 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).

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.

The SHA-256, SHA-384, SHA-512, and SHA-512/256 hash functions meet the 128-bit strength requirements for collision resistance and second preimage resistance[6]. This leads to the following guidelines:

GEN-19. [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-20. [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-21. 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.

[5] 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.

[6] See FIPS 180-4 [FIPS180-4].

4.6.3 Handling of Path Separators and Environment Variables

The third issue to be addressed here concerns interoperable handling of path separators and environment variables in <File> and <Directory> elements.

The SWID specification defines three attributes which may be used on <File> and <Directory> elements to fully specify a file or a folder:

- location: A string specifying the directory or location where a file was found or can be expected to be located
- name: A string specifying the name of the filename or directory without any embedded path separators
- root: A string specifying a system-specific root folder that the 'location' attribute is an offset from; if this is not specified, it is assumed that the 'root' is the same folder as the location of the SWID tag, or is the directory specified by an enclosing <Directory> element

While the 'name' attribute is defined to explicitly exclude path separators, both 'location' and 'root' are permitted to include such separators. The problem is that path separators vary across operating environments. The most widely known difference is between Windows and UNIX/Linux systems; Windows uses the backslash '\' as the path separator, while UNIX/Linux systems use the forward slash '/'.

It is important that tag consumers be able to reliably parse strings containing embedded path separators without having to guess the path separator. This document recommends that the path separator character be made explicit in <Payload> and <Evidence> elements. The following guideline achieves this by introducing a new @n8060:pathSeparator extension attribute:

> **GEN-22.** The @n8060:pathSeparator extension attribute SHOULD be used within <Payload> and <Evidence> elements to specify the path separator character used in embedded <File> and <Directory> elements.

In a related vein, the SWID specification also allows platform-specific environment variables to be used within any or all of the 'root', 'location', and 'name' attributes. Once again, the format of such variables differs across platforms. On Windows machines, environment variables are enclosed in percent '%' characters, while UNIX/Linux simply prefix variables with a dollar '$' character. This document recommends that environment variable prefix and suffix characters be made explicit in <Payload> and <Evidence> elements. The following guidelines achieve this by introducing new @n8060:envVarPrefix and @n8060:envVarSuffix extension attributes:

> **GEN-23.** The @n8060:envVarPrefix extension attribute SHOULD be used within <Payload> and <Evidence> elements to specify the character(s) used to prefix environment variables that may be embedded <File> and <Directory> elements.

> **GEN-24.** The @n8060:envVarSuffix extension attribute SHOULD be used within

<Payload> and <Evidence> elements to specify the character(s) used to suffix environment variables that may be embedded <File> and <Directory> elements.

When guidelines **GEN-22** through **GEN-24** are observed, a <Payload> could be represented as follows:

```
<Payload n8060:pathSeparator="/" n8060:envVarPrefix="$"
    n8060:envVarSuffix="">
  <Directory root="$ETC/drivers" name="printers">
    <Directory name="printermodel">
      <File name="colordriver.shlib" size="234824"/>
      <File name="bwdriver.shlib" size="143854"/>
    </Directory>
  </Directory>
</Payload>
```

Given this information, a tag consumer should be able to straightforwardly construct these two fully-qualified filenames:

- $ETC/drivers/printers/printermodel/colordriver.shlib
- $ETC/drivers/printers/printermodel/bwdriver.shlib

To access these files on the device, device-specific information will still be required to determine the value of the '$ETC' environment variable.

4.7 Providing Attribute Values in Multiple Languages

Section 4.3 introduced a guideline to require that the <SoftwareIdentity> @xml:lang attribute be used to specify the preferred human language used within the tag to provide language-dependent attribute values. This is the most common scenario, i.e., that all language-dependent attribute values within a tag will be provided in the same language. Such tags will be termed *monolingual tags*. For example, a Japanese software provider wishing to create a monolingual tag in Japanese will set the preferred language of their tag to be Japanese (e.g., using the language tag "ja-jp" as the value for the <SoftwareIdentity> @xml:lang attribute), then specify all language-dependent attributes in Japanese.

Because the SWID specification permits the @xml:lang attribute be used on *any* tag element, this enables tag creators to implement *multilingual tags*, tags which provide language-dependent attributes in more than one language. Suppose, for example, that a Japanese software provider wants to provide their organization's name in both Japanese and English. Technically, this is straightforward to do, as illustrated here:

```
<SoftwareIdentity
  xmlns="http://standards.iso.org/iso/19770/-2/2015/schema.xsd"
  xml:lang="ja-jp"
  name="コンピュータ管理システム 2015"
  tagId="jp.largecomputerco.タグ番号 1"
  version="1.0">
```

```
  <Entity
    name="大型コンピュータ会社"
    regid="largecomputerco.jp"
    role="tagCreator softwareCreator"/>
  <Entity xml:lang="en-us"
    name="Large Computer Company"
    regid="largecomputerco.jp"
    role="tagCreator softwareCreator"/>
  …
</SoftwareIdentity>
```

The first <Entity> element in the example inherits the value of @xml:lang specified by the <SoftwareIdentity> element. The second <Entity> element explicitly overrides the preferred language, signaling that information is being furnished in (in this example) English.

Although the SWID specification provides the technical means to implement multilingual tags, in practice, implementing and processing such tags present a number of challenges and potential interoperability issues, so care must be taken.

This report does not offer guidelines to address these issues, since the marketplace requirements for multilingual tags are insufficiently clear to support development of robust guidelines to fully and effectively address all associated interoperability concerns. Instead, this report will discuss some of the most significant issues and suggest future directions for interoperability guidance.

4.7.1 Specifying Product Names in Multiple Languages

The <SoftwareIdentity> @name attribute specifies the preferred name of the software product, and is provided in the language indicated by the value of the @xml:lang attribute (which is now required per guideline **GEN-2**). The SWID specification makes no provisions for the specification of multiple product names, much less for the specification of multiple such names in different languages.

If the marketplace determined that there is a demand for recording a multilingual representation of a product name in a single SWID tag, two options (short of revising the SWID specification itself) would seem to exist:

1. Use the <Meta> element. Note, however, that because the <Meta> element described in the SWID specification does not offer a predefined attribute for alternate product names, the user community would need to agree on a new extension attribute to be used for this purpose.
2. Use supplemental tags. Here, the idea would be to create one or more supplemental tags, each of which specifies a different preferred language via the <SoftwareIdentity> @xml:lang attribute, that then provides the <SoftwareIdentity> @name attribute value in that preferred language. In other words, a U.S.-based software provider might issue a single primary tag providing the product name in English, along with four supplemental tags providing the product name in each of French, German, Japanese, and Spanish.

The first option is relatively compact but depends on effective procedures for defining and managing extension attributes. The second option avoids the introduction of an extension attribute, but forces tag creators to generate arbitrary quantities of supplemental tags to address all human languages of interest.

4.7.2 Specifying <Entity> Elements in Multiple Languages

The example at the top of this section shows how a Japanese organization's name could be provided in both Japanese and English, in two separate `<Entity>` elements distinguished by different `@xml:lang` attributes, one value inherited from `<SoftwareIdentity>` and the other an explicit value overriding the inherited value.

This multilingual capability also creates opportunities for entity information specified in one language to differ in unexpected and confusing ways from entity information specified in a second language. For example, there is nothing in either the SWID specification or the associated XML schema that would prevent the following scenario:

```
<SoftwareIdentity
  xmlns="http://standards.iso.org/iso/19770/-2/2015/schema.xsd"
  xml:lang="ja"
  name="コンピュータ管理システム 2015"
  tagId="jp.largecomputerco.タグ番号 1"
  version="1.0">
  <Entity xml:lang="en"
    name="The Large Computer Company"
    regid="largecomputerco.jp"
    role="tagCreator softwareCreator"/>
  <Entity xml:lang="fr"
    name="Le Grand Enterprise d'Informatique"
    regid="largecomputerco.jp"
    role="tagCreator"/>
  …
</SoftwareIdentity>
```

This example contains two problems. First, the tag's preferred language is specified as Japanese, but entity information is only provided in English and French. Given such a tag, it is not obvious how a tag consumer should decide which `<Entity>` element provides the preferred name of the tag creator. Second, the two `<Entity>` elements agree on the `@regid`, but disagree on the `@role`. Again, it is not obvious how to interpret such data. A number of equally odd scenarios can easily be envisaged.

Future guidelines may be needed to address issues like these. Such guidelines might include:

- A requirement that at least one `<Entity>` element be provided in the preferred language
- A requirement that all `<Entity>` elements which agree on the `@regid` value also agree on the `@role` value, irrespective of the `@xml:lang` value

4.7.3 Specifying <Payload> Elements in Multiple Languages

When software products are localized to a particular language region, files and folder names often change to match the localized language. To account for this, authoritative tag creators may want to create multilingual tags that provide `<Payload>` elements which enumerate alternate language-specific versions of files and folders. If this is done, it must be done in a way that enables tag consumers to simply and accurately determine the effective language-specific payload. Note that non-authoritative tag creators are assumed to be unable to produce tag data in multiple languages, so this issue is not a concern for `<Evidence>` elements.

This information could be represented in multiple ways; for example, a tag creator could implement multiple alternative `<Payload>` elements as illustrated here:

```
<SoftwareIdentity
  xmlns="http://standards.iso.org/iso/19770/-2/2015/schema.xsd"
  xml:lang="en"
  name="Joyful App 2015"
  tagId="com.largecomputerco.joyfulapp1"
  version="1.0">
  …
  <Payload
    <File name="joyfulapp.exe"/>
  </Payload>
  <Payload xml:lang="fr"
    <File name="appdejoie.exe"/>
  </Payload>
  …
</SoftwareIdentity>
```

In this example, the first `<Payload>` element is expressed in the tag's preferred language (English), inherited from `<SoftwareIdentity>`. The second `<Payload>` element is expressed in French. This is a simple and straightforward way to represent equivalent payloads in alternate languages. But because `@xml:lang` may be used on any element, there is nothing to prevent a tag creator from representing the payload in the following way:

```
  <Payload
    <File name="joyfulapp.exe"/>
    <File xml:lang="fr" name="appdejoie.exe"/>
  </Payload>
```

Arguably, this is a more compact representation, but raises questions of how one might determine the effective language-specific payload for a given device. The above payload could potentially represent two files that are co-present on a device, or two different files, only one of which is present as determined by the local language. Future guidelines may be necessary to resolve potential interoperability issues like these.

4.8 Updating Tags

Although the SWID specification does not prohibit modification of SWID tags, it does restrict modifications so that they can only be performed by the original tag creator. The primary reason for altering a tag after it has been installed on a device is to correct errors in the tag. In rare circumstances, it may be useful to update a tag to add data elements that logically belong in the tag and not in a separate supplemental tag. However, 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 along with any embedded digital signatures.

When a tag must be updated to correct errors or add data elements, its `<SoftwareIdentity>` `@tagId` should not be changed. This is because tag identifiers may be used as proxy 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-25.** 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 compare all contents of discoverable tags to determine if any of the products have changed since the last time the tags were examined. To facilitate change detection by tag consumers, tag creators are expected to update the `<SoftwareIdentity>` `@tagVersion` attribute to indicate that a change has been made to the tag.

> **GEN-26.** 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.

4.9 Summary

The primary purpose of all the guidelines 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 guidelines are intended to be broadly applicable to common IT usage scenarios that are relevant to all software providers and consumers.

This section provided implementation guidelines addressing issues common to all situations in which tags are deployed and processed. These are the key points from this section:

- Tags may be created by authoritative or non-authoritative entities. An *authoritative tag creator* is a first- or second-party that creates a tag as part of the process of releasing software. Authoritative tag creators typically possess accurate, complete, and detailed technical knowledge that is needed for creation of authoritative tags containing reliable information. A *non-authoritative tag creator* is an entity 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 forensic methods while discovering installed software.
- The SWID specification supports an international audience, allowing tag creators to provide language-dependent attribute values in region-specific human languages. Guidelines in this section specified how tag creators should designate the default human language of language-dependent attribute values provided within a tag, and how such values may be provided in multiple languages.
- SWID tags provide detailed information about various entities associated with the software product described the tag, as well as with the tag itself. Guidelines in this section addressed how complex entity specifications are to be avoided, how authoritative and non-authoritative tags are to be distinguished, and how information about the software creator is to be furnished.
- SWID tags may be explicitly linked to other tags and/or other resources in a variety of ways. Guidelines in this section addressed how source tags are to be linked to individual target tags as well and/or to sets of target tags.
- Tag creators may provide detailed information about the files and folders comprising a software product. Guidelines in this section addressed how sufficient information may be provided, how cryptographic hashes may be provided, and how platform-specific path separators and environment variables may incorporated in file or folder descriptions.

5 Implementation Guidance Specific to Tag Type

This section provides implementation guidelines that are specific for each of the four tag types defined in Section 2.1: corpus tags (see §5.1), primary tags (see §5.2), patch tags (see §5.3), and supplemental tags (see §5.4).

5.1 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 (see §5.1.1), specifying `@version` and `@versionScheme` (see §5.1.2), and specifying `<Payload>` element information (see §5.1.3).

5.1.1 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`".

5.1.2 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`. Table 3 lists the allowed values of `@versionScheme` that are defined in the SWID specification.

Table 3: Allowed Values of @versionScheme

Value	Meaning
`multipartnumeric`	Numbers separated by dots, where the numbers are interpreted as integers (e.g. 1.2.3, 1.4.5, 1.2.3.4.5.6.7)
`multipartnumeric+suffix`	Numbers separated by dots, where the numbers are interpreted as integers with an additional string suffix (i.e. 1.2.3a)
`alphanumeric`	Strictly a string, sorting is done alphanumerically
`decimal`	A floating point number (e.g. 1.25 is less than 1.3)
`semver`	Follows the [SEMVER] specification.
`unknown`	Other unknown version scheme, no attempt should be made to order versions of this type
`<any>`	Other version schemes that may be generally known in the market

The following guideline is provided in consideration of the fact that tag consumers have a critical interest in knowing not only a product's version, but also its versioning scheme and the semantics of that scheme.

> **COR-3.** If a corpus tag contains a value for the `<SoftwareIdentity>` `@version` attribute, it MUST also contain a value for the `<SoftwareIdentity>` `@versionScheme` attribute.

If a particular product's version does not conform to one of the pre-defined schemes listed in Table 3, whatever value a tag creator provides for the `<SoftwareIdentity>` `@versionScheme` attribute ought to be selected from a well-known public list of version scheme identifiers. Mechanisms for establishing, advertising, and curating such public lists are beyond the scope of this document. Ideally, such well-known public lists of version schemes will provide enough semantic definition of each scheme to enable tag consumers to determine

whether a version V1 conforming to a particular scheme should be ordered 'before' or 'after' another version V2 conforming to that same scheme.

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

5.2 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 (see §5.2.1), specifying version and version scheme information (see §5.2.2), specifying <Payload> or <Evidence> information (see §5.2.3), and specifying attributes needed to form Common Platform Enumeration (CPE) names (see §5.2.4).

5.2.1 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".

5.2.2 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, and that version can be determined, the `<SoftwareIdentity>` `@version` attribute of the primary tag MUST contain that value.

PRI-5. [Non-Auth] If a primary tag contains a value for the `<SoftwareIdentity>` `@version` attribute, and the version scheme of that `@version` attribute value can be determined, the `<SoftwareIdentity>` `@versionScheme` attribute of the primary tag MUST contain that version scheme value.

As was true for corpus tags (see §5.1.2), it is important that the version schemes used in primary tags enable distinct versions of a product to be placed in a defined order, minimally so that consumers can determine whether one version of a product is 'before' (earlier) or 'after' (later) than another version. Section 8.6.13 of the SWID specification provides a table of predefined values for the `@versionScheme` attribute with defined semantics (reproduced above in Table 3). If a value for the `@versionScheme` attribute is provided that is not listed among the predefined values, ideally that value ought to come from a well-known public list of version scheme identifiers. The public list would specify the meaning for each version scheme sufficiently to allow for comparing two versions and determining their relative order in a sequence of versions.

5.2.3 Specifying Primary Tag Payload and Evidence

Detailed information about the files comprising an installed software product is a critical need for cybersecurity operations. For example, 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. Authoritative tag creators are encouraged to provide a `<Payload>` element in the primary tag, and non-authoritative tag creators are encouraged to provide an `<Evidence>` element in the primary tag.

PRI-6. [Auth] A `<Payload>` element SHOULD be provided in a software product's primary tag.

Note:Payload information from authoritative tag creators is a key enabler for a number of cybersecurity usage scenarios, and promises to dramatically increase the value of SWID tags to tag consumers. At this time, however, a weaker guideline is presented until the potential costs and burdens on tag creators and consumers can be better understood, along with a better sense of the methods needed for providing this information in a way that supports the appropriate level of assurance.

PRI-7. [Non-Auth] An `<Evidence>` element SHOULD be provided in a software product's primary tag.

Note : Guidelines **PRI-6** and **PRI-7** currently specify that payload and evidence be supplied within the primary tag, and not within a supplemental tag. This is due to concerns about additional processing complexity and difficulties with assuring the reliability of such payload and evidence information when it is stored separately from the primary tag. As the degree of understanding of payload and evidence usage patterns improves, providing a clearer sense of the costs and benefits, strengthening or refinement of these requirements may be needed.

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. It also aids in reconciling authoritative and non-authoritative tags in cases where both kinds of tags exist on a device for the same product.

PRI-8. <Payload> and <Evidence> elements SHOULD list every file comprising the product described by the tag.

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

PRI-9. [Auth] The <Payload> element MUST list every machine instruction file comprising the product described by the tag.

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

If a tag creator enumerates every file according to **PRI-8**, this can cause problems later for tag consumers. Recall that Section 4.6 of this document provides guidelines that require tag creators to supply file size, version, and hash information (see guidelines **GEN-11** through **GEN-18**). These guidelines are there to ensure that tag consumers can later use the provided information to confirm the integrity of files discovered on devices. The problem, however, is that particular files listed in a <Payload> or <Evidence> element might be changed for non-malicious reasons at arbitrary times after the product is installed. Data and configuration files are two obvious examples.

If a tag consumer were to inspect a particular file listed in a tag's <Payload> or <Evidence> element, compare the file's hash value as listed in a tag to a new value computed from the actual file on a device, and discover a mismatch, that would be a "false positive" if that file were not expected to be static. If tag consumers were to generally find that performing such comparisons led to an unwieldy number of false positives, they might be inclined to stop using SWID tag payload and evidence information altogether, an undesirable outcome.

In the interest of minimizing the possibility of such false positives, this document provides guidelines for tag creators to explicitly mark all mutable files listed in a tag's <Payload> element with a special value. Specifically, this document introduces a @n8060:mutable extension attribute on <File> elements. The @n8060:mutable extension attribute takes a Boolean value whose default value is "false". Authoritative tag creators are *required* to set the @n8060:mutable attribute value to "true" for any <File> element that describes a non-static file. Non-authoritative tag creators are encouraged to do so whenever possible.

PRI-11. [Auth] If a <File> element included in a <Payload> element of a primary tag describes a file that can undergo authorized changes over time in ways that could alter its size, version, and/or hash value, the tag creator MUST set that file's <File> @n8060:mutable extension attribute to "true".

PRI-12. [Non-Auth] If it can be determined that a <File> element included in an <Evidence> element of a primary tag describes a file that can undergo authorized changes over time in ways that could alter its size, version, and/or hash value, the tag creator SHOULD set that file's <File> @n8060:mutable extension attribute to "true".

Observance of these guidelines by tag creators will help ensure that the resulting <Payload> and <Evidence> elements are useful to tag consumers attempting to verify the integrity of installed software products, while minimizing the potential number of false positives that such consumers may have to cope with.

Note: Late in the process of readying Draft #4 of this report for public comment, it was noted that the guidelines in this section also apply to corpus and patch tags. In the final draft of this report, these guidelines will be incorporated into Section 4.6. They are left here for now in order to minimize impact to the extensive guideline cross-references throughout this report in the current draft revision.

5.2.4 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 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 [NISTIR 8085] 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.

5.3 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 (see §5.3.1), linking patch tags to related tags (see §5.3.2), and specifying <Payload> or <Evidence> information (see §5.3.3).

5.3.1 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".

5.3.2 Linking Patch Tags to Related Tags

A patch tag must be explicitly linked to the primary tag for the product to which the patch is applied. The SWID specification defines three relations which may be used when setting the value of the <Link> @rel attribute. These relations are described in Section 5.3.3 of the SWID specification and summarized in Section 2.1.3 of this report. They are:

- **Patches.** This value documents a relationship to the primary tag of the patched product.
- **Requires.** This value documents that a patch described by the patch tag requires the prior installation of another patch.
- **Supersedes.** This value documents that a patch described by a patch tag can entirely replace another patch.

Because the SWID specification requires that patch tags use these relations as appropriate when linking to related tags, no additional guidelines are provided in this report.

5.3.3 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-14** and **GEN-16**, 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 (see **GEN-14, GEN-17**).

PAT-2. [Auth] A patch tag MUST contain a `<Payload>` element that MUST enumerate every file that is modified, removed, or added by the patch.

PAT-3. [Auth] Each `<File>` element contained within the `<Payload>` element of a patch tag MUST include an extension attribute named `@n8060:patchEvent`, which MUST have one of the following values:

- The string value "`modify`" to indicates that the patch modifies a pre-existing installed file
- The string value "`remove`" to indicates that the patch removes a pre-existing installed file
- The string value "`add`" to indicates that the patch installs a new file that did not previously exist

PAT-4. [Non-Auth] A patch tag MUST contain an `<Evidence>` element that enumerates every file that was found to have changed as a result of the patch process.

5.4 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 (see §5.4.1), linking supplemental tags to other tags (see §5.4.2), and establishing the precedence of information contained in a supplemental tag (see §5.4.3).

5.4.1 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`".

5.4.2 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.

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`".

Note that the SWID specification also requires that every `<Link>` element provide a value for the `@href` attribute. Section 4.5 of this document provides pertinent guidelines for how tag creators should use the `@href` attribute to refer to other tags, in situations when the `@tagId` of the target is known (see guideline **GEN-11**), and when it is not known (see guidelines **GEN-12** and **GEN-13**).

5.4.3 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 tag being supplemented, the data value in the supplemented tag MUST be considered to be the correct value.

5.5 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 metadata-based searching and 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.

6 SWID Tag Usage Scenarios

This section presents a set of usage scenarios (US) that illustrate how, based on the guidelines provided in Sections 4 and 5 of this document, security professionals can achieve three important cybersecurity objectives:

1. Minimizing exposure to publicly disclosed software vulnerabilities (see §6.1)
2. Enforcing organizational policies regarding authorized software (see §6.2)
3. Controlling network resource access from potentially vulnerable endpoints (see §6.3)

By using SWID tags in accordance with the guidelines provided in this report, the security practitioner (e.g., Chief Information Security Officer (CISO), Information System Security Officer (ISSO)) can achieve these objectives quickly, accurately, and efficiently. Sections 6.1 through 6.3 each describe the cybersecurity objective to be achieved, followed by specific usage scenarios that contribute to achieving the objective. Section 6.4 describes how the guidelines presented in this report enable each scenario.

6.1 Minimizing Exposure to Publicly-Disclosed Software Vulnerabilities

This section presents usage scenarios illustrating how SWID tags may be used by security practitioners to minimize risks from exploitation of endpoints with known vulnerabilities within enterprise networks. To minimize these risks, security practitioners need to maintain awareness of vulnerabilities related to installed software, especially those vulnerabilities for which a patch or other remediation has not been made available. Security practitioners also need to maintain awareness of changes to the software inventory 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 mitigates a critical vulnerability.

This section presents four usage scenarios related to this cybersecurity objective:

- US 1 – Continuously Monitoring Software Inventory (see §6.1.1)
- US 2 – Ensuring that Products are Properly Patched (see §6.1.2)
- US 3 – Correlating Inventory Data with Vulnerability Data to Identify Vulnerable Endpoints (see §6.1.3)
- US 4 – Discovering Vulnerabilities Due to Orphaned Software Components (see §6.1.4)

6.1.1 US 1 – Continuously Monitoring Software Inventory

In this scenario, SWID tags are used to continuously monitor the inventory of software installed on endpoints within an enterprise network. Tags are key inputs to the process of gathering and maintaining an up-to-date and accurate accounting of software inventory on each endpoint. SWID data may be aggregated, if needed, in regional and/or enterprise-wide repositories. Using this data, organizations are able to maintain an ongoing understanding of installed software inventory by continuously monitoring software-change event notifications. 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 discovery or monitoring tools (referred to generically in this section as "discovery tools") 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 §2.3.1 for a discussion of SWID tag placement on devices), including these:

- 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 drives, Universal Serial Bus (USB) devices)
- Software contained in native package installers (e.g., RPM Package Manager (RPM))
- Shared filesystems (e.g., a mapped network drive or network-attached storage) that contain software that is executable from an endpoint.

SWID tags provide identification, metadata, and relationship information about an endpoint's installed software. Authoritative tags discovered on an endpoint can supply reliable and comprehensive information about installed software, whereas discovery tools can place non-authoritative SWID tags on the endpoint to leave a record of newly-discovered, untagged products. 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 analysis and 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 to monitor endpoints for the purpose of maintaining continual awareness of their security posture.

6.1.1.1 Initial Conditions

This usage scenario assumes the following conditions:

- A software discovery tool is installed on each enterprise-managed endpoint, and is configured to run on a defined schedule, on request, and/or in response to events generated on the endpoint which may indicate there has been a change to the installed software inventory.
- The discovery tool records inventory data in a configuration management database (CMDB) which may or may not be co-resident on the endpoint.
- The CMDB retains information about products (and their associated tags, if any) which have been discovered in the past on each endpoint. The discovery tool is able to use the

CMDB to compare current inventory state to the last known state, in order to detect changes.

- At the time the discovery process is run on each endpoint, the discovery tool has sufficient access rights to the endpoint to discover each installed software instance and any associated metadata. This includes access rights to read SWID tags on the endpoint.
- Some installed software products might not have an associated SWID tag because an authoritative source did not furnish one.

6.1.1.2 Process

1. Upon detecting new or changed software in an installation location or in a filesystem mounted on the endpoint, the discovery tool will collect and process all SWID tags (primary, supplemental, and/or patch tags) present in that location. Changes to be detected may include:
   - New software products (or subcomponents) that were not previously recorded in the inventory
   - Changes or updates to installed software products discovered previously
   - New or modified SWID tags, as indicated by a new `@tagId` or `@tagVersion` attribute value within the `<SoftwareIdentity>` element
2. The discovery tool will update the CMDB with the data from newly discovered or changed SWID tags, creating new entries and/or modifying existing entries as needed for installed 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 (see **PRI-2, PRI-3, PRI-4, PRI-5**).

   If any tags are identified as not being in compliance with the SWID specification (see **GEN-1**), those tags will not be recorded in the CMDB, 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 (see **PRI-1, COR-1, PAT-1, SUP-1**). 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 (see **GEN-14, GEN-15, GEN-16, GEN-17, GEN-18, GEN-19, GEN-20, GEN-21, GEN-22, GEN-23, GEN-24, PRI-6, PRI-7, PRI-8, PRI-9, PRI-10, PRI-11, PRI-12)**.
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`" (see §4.4.3, **GEN-8, GEN-9, GEN-10**).
5. If a tag was not installed with the software, the discovery tool will create and deploy a non-authoritative tag to the endpoint for each instance of a discovered application. As an

alternative, the discovery tool may be able to determine that a previously generated non-authoritative tag already exists in the CMDB which describes the newly-discovered product. This is possible if (1) the CMDB records all tags—authoritative and non-authoritative—discovered or deployed anywhere within the enterprise, and (2) the discovery tool is able to use information about the newly-discovered product to retrieve a matching tag from the CMDB. In this case, the discovery tool may simply deploy the matching tag from the CMDB (or an appropriate related source) to the endpoint, rather than generate and deploy a completely new non-authoritative tag.

Information about the files discovered is important to support continuous monitoring for software vulnerabilities, so the deployed tag will list every machine instruction file comprising the software product discovered (See §5.2.3), using the `<Evidence>` element (see **PRI-7, PRI-8**). This information will include filenames, sizes, versions, and cryptographic hashes discovered (see **GEN-14, GEN-15, GEN-17, GEN-18, GEN-20, GEN-21, GEN-22, GEN-23, GEN-24**). It will also include any version information determined for the software product (see **PRI-4, PRI-5).**

When SWID tags are discovered that do not conform to the 2015 release of the SWID specification, these tags are not stored in the CMDB, but their contents might still 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 authenticate the certificate in the digital signature using the `@thumbprint` attribute of the `<Entity>` element (see §3.2).

7. The discovery tool will read the tag identifier (i.e., `@tagId`) and identify the tag location, along with the type of tag discovered or created: primary tags for installed software (see §2.1.2**)**, and patch tags for software patches (see §2.1.3). Supplemental tags can provide additional information and may be useful for inventory. 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 (see **GEN-25, GEN-26**).

8. The CMDB will be updated, including sending notifications to applicable reporting systems in the enterprise. The CMDB 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 (see **GEN-11, GEN-12, GEN-13**). 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 (see **SUP-2**). If any data in the supplemental tag conflicts with the data in any tag it supplements, the data in the supplemented tag is considered the correct value (see **SUP-3**).

6.1.1.3 Outcomes

The process described above provides an accurate and automated method for collecting identifying and descriptive metadata about an endpoint's inventory of 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 software 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 Sections 3.1.1 and 3.1.5.

As an indirect result of maintaining SWID tag-based inventory, the discovery tools can dynamically identify vulnerabilities and misconfiguration. For example, upon discovering a newly installed or changed software application, the discovery tool can check the configuration of that software using a pre-defined checklist. The discovery tool could also check for any known vulnerabilities for that new or updated product. If the tool identifies a misconfiguration or a software vulnerability, that condition may be reported for mitigation.

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.

6.1.2 US 2 - Ensuring that Products Are Properly Patched

Enterprise security managers often need to quickly and easily generate reports about endpoints having installed software products that are missing one or more patches, as this may signal vulnerability to malicious activity. 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 individual endpoint can also directly read the inventory of patch tags from the local repository to enable localized patch verification and decision making.

6.1.2.1 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 (see §6.1.1). This includes application of guidelines **GEN-1** through **GEN-26**.

6.1.2.2 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 whether any other required predecessor patches are also present. This is done by inspecting embedded `<Link>` elements where the `@role` attribute value is "`requires`" (Per §5.3.3 of the SWID specification), then confirming the presence of the target tag. 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 §5.3 (see **PAT-2, PAT-3, PAT-4**) 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 `@n8060: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 §5.3.3 of the SWID specification, to see if that patch has been superseded by a newer patch. 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.

6.1.2.3 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 by taking advantage of `<Payload>` or `<Evidence>` elements contained in patch tags, 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.

6.1.3 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.

Because SWID tags adhere to a consistent and standardized structure, they aid automated correlation of information published by vulnerability information sources (e.g., NIST's National Vulnerability Database, US-CERT alerts, as well as vulnerability advisories issued by vendors and independent security analysts) 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]. [NISTIR 8085] 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 (see §5.2.4), 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.

6.1.3.1 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 (see §6.1.1). This includes application of guidelines **GEN-1** through **GEN-26**.

6.1.3.2 Process

1. Using product advisories containing information about publicly disclosed 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 primary tag `<SoftwareIdentity>` `@name` and `@version` (see **PRI-2**, **PRI-3, PRI-4, PRI-5**), and `<Meta>` `@revision` and `@edition` (see **PRI-13**). Additionally, by forming CPE names from SWID tags (see [NISTIR 8085]), the discovery tool can search for endpoints with software referenced by those CPE names included in the vulnerability bulletins.

2. If the bulletin references the tag identifier for the relevant tag for a software product or patch, the discovery tool will search for that identifier. SWID tags adhering to guidelines **PRI-1** through **PRI-5** 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-6** 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",[7] 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 (see US 2, §6.1.2), including checks for predecessor or superseded patches.

6. If the endpoint is found to contain vulnerable software but not the associated patch(es), the endpoint may be flagged as potentially in need of remediation activities.

[7] More information about the Heartbleed bug is available from www.heartbleed.com

Consider the case of a fictional vulnerability involving a known buffer overflow in the product named Acme Roadrunner, affecting versions between 11.1 and 11.7 and between 12.0 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.

6.1.3.3 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.

6.1.4 US 4 - Discovering Vulnerabilities Due to Orphaned Software Components

Orphaned software is a component that is left behind on an endpoint after all software products which require it are uninstalled. Components which frequently become orphaned software include shared library files, software patches, and configuration files. If an orphaned component contains an exploitable flaw and can be caused to execute, its presence can make an endpoint vulnerable to attach. Additionally, orphaned software wastes 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 (see §3.1.4) enables understanding of how software components relate, supporting software asset management decisions.

6.1.4.1 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 (see §6.1.1). This includes application of guidelines **GEN-1** through **GEN-26**.

- 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 §5.3.3 of the SWID specification.

6.1.4.2 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 not 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 may be able to be removed as well.

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

6.1.4.3 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.

6.2 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 presents two usage scenarios:

- US 5 – Preventing Installation of Unauthorized or Corrupted Software Products (see §6.2.1)
- US 6 – Discovering Corrupted Software and Preventing Its Execution (see §6.2.2)

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

To strictly control what software may or may not be installed on enterprise endpoints, corpus tags may be used whenever a software installation is initiated to confirm that the software to be installed either is included in a product whitelist, or is not included in a product 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., "public-facing webserver") might have unique lists of allowed or prohibited software.

As described in §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 §3.1.6 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.

6.2.1.1 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.
- The software installation system is policy-aware, in that it is able to consult organization-specific whitelists/blacklists for the purpose of restricting software installations only to allowed products.
- If the issuer of the installation package is to be verified, the corpus tag must be digitally signed.

6.2.1.2 Process

1. Upon initiation of a software installation on an endpoint, the policy-aware installation system discovers a corpus tag included with the distribution package of the software product whose installation has been requested. If no corpus tag is found, some installation systems may be configured by enterprise policy to prevent the installation, 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 it to a 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 may be prevented, ending this process.

3. The installation tool determines whether the `@tagId` or other data included in the tag (see **COR-1** through **COR-4)** 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 (see **COR-4**, **GEN-14, GEN-15, GEN-16, GEN-19, GEN-21**). If any hash is found not to match, the installation is prevented, ending this process.

6.2.1.3 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.

6.2.2 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 compromised 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 actual observed on the endpoint.

Detection of software 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 unauthorized alteration

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 §4.6.2 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.

6.2.2.1 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 (see §6.1.1). This includes application of guidelines **GEN-1** through **GEN-26**.

6.2.2.2 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-6** 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 §4.6.2.

   Similarly, the discovery tool examines each patch tag to gather the cryptographic evidence for files added or changed by that patch (see **PAT-2, PAT-3, PAT-4**).

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 reports the variance.

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

6.2.2.3 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 without authorization.

6.3 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 and 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., missing patches).

In cases where a discovery tool is able to detect that an endpoint is providing inventory data that conflicts with known information, that condition may indicate that malicious software on that endpoint is intentionally misrepresenting the endpoint's condition. Such a conflict might provide an indicator of compromise that could trigger further investigation.

6.3.1 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 (see §6.1.1). This includes application of guidelines **GEN-1** through **GEN-26**.

6.3.2 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 (see §6.1.1)
   - If an endpoint is properly patched (see §6.1.2)
   - If the endpoint is in compliance with whitelist or blacklist requirements (see §6.2.1)

6.3.3 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.

6.4 Association of Usage Scenarios with Guidelines

To aid in the association of usage scenarios to the guidelines described in Sections 4 and 5, 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 4 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 as described in the ISO 19770-2:**2015** specification so that they are interoperable among discovery tools (see **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 (see **COR-1, PRI-1, PAT-1, SUP-1**) to aid in achieving SAM objectives and attaining cybersecurity goals.
- **Internationalization –** To support an international audience, tag creators are able to provide language-dependent attribute values in region-specific human languages (see **GEN-2**, **GEN-3**).
- **Tag Entities** – SWID tag creators must concisely and accurately record entities, which have an important role in addressing potential interoperability issues that could arise when tag creators specify attribute values in multiple human languages (see **GEN-4** through **GEN-7**).
- **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 (see **GEN-8** through **GEN-10**).
- **Tag Linking** – Discovery tools need consistent reference information to be able to link a source tag to one or more target tags (see **GEN-11** through **GEN-13**).
- **File Information** – Discovery tools use the files' names, sizes, and cryptographic hashes included in SWID tags to validate integrity (see **GEN-14** through **GEN-24)**.
- **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 (see **GEN-25**, **GEN-26**).
- **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 (see **COR-2**, **COR-3**).
- **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 (see **COR-4)**.
- **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 (see **PRI-2** through **PRI-5**).
- **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>` or `<Evidence>` information included in the SWID tag (see **PRI-6** 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 helps accurately identify products and components installed (see **PRI-13**).
- **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**.
- **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 4 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 4: Relationship of Guidelines to Usage Scenarios

Guideline Family	Guideline	Usage Scenarios						
		US 1	US2	US3	US4	US5	US6	**US7**
Basic Compliance	GEN-1	●	●	●	●	●	●	●
Tag Type	COR-1	●	●	●	●	●	●	●
	PRI-1	●	●	●	●		●	●
	PAT-1	●	●	●	●		●	●
	SUP-1	●	●	●	●		●	●
Internationalization	GEN-2	●	●	●	●		●	●
	GEN-3	◒	◒	◒	◒		◒	◒
Tag Entities	GEN-4	●	●	●	●		●	●
	GEN-5	◒	◒	◒	◒		◒	◒
	GEN-6	●	●	●	●		●	●
	GEN-7	●	●	●	●		●	●
Tag Authority	GEN-8	●	●	●	●		●	●
	GEN-9	●	●	●	●		●	●
	GEN-10	◒	◒	◒	◒		◒	◒
Tag Linking	GEN-11	●	●	●	●		●	●
	GEN-12	●	●	●	●		●	●
	GEN-13	●	●	●	●		●	●

Guideline Family	Guideline	Usage Scenarios						
		US 1	US2	US3	US4	US5	US6	**US7**
File Information	GEN-14	●	●	●	●	●	●	●
	GEN-15	●	●	●	●	●	●	●
	GEN-16	●	●	●	●	●	●	●
	GEN-17	◒	◒	◒	◒		◒	◒
	GEN-18	◒	◒	◒	◒	◒	◒	◒
	GEN-19	●	●	●	●	●	●	●
	GEN-20	◒	◒	◒	◒		◒	◒
	GEN-21	◒	◒	◒	◒	◒	◒	◒
	GEN-22	◒	◒	◒	◒		◒	◒
	GEN-23	◒	◒	◒	◒		◒	◒
	GEN-24	◒	◒	◒	◒		◒	◒
Tag Updates	GEN-25	◒	◒	◒	◒		◒	◒
	GEN-26	●	●	●	●		●	●
Pre-Installation Software Version	COR-2					●		
	COR-3					●		
Media Integrity Verification	COR-4					●		
Post-Installation Software Version	PRI-2[A]	●		●				
	PRI-3[A]	●		●				
	PRI-4[N]	●		◒				
	PRI-5[N]	●		◒				
Software Integrity Verification	PRI-6[A]	●		●			●	
	PRI-7[N]	◒		◒			◒	
	PRI-8	◒		◒			◒	
	PRI-9[A]	●		●			●	
	PRI-10[N]	●		●			●	
	PRI-11[A]	●		●			●	
	PRI-12[N]	◒		◒			◒	
Metadata	PRI-13			●				
Patch Relationships	PAT-2[A]		●				●	
	PAT-3[A]		●				●	
	PAT-4[N]		●				●	
Supplemental Tag Relationships	SUP-2	●						
Data Deconfliction	SUP-3	●						
● indicates mandatory guideline, ◒ indicates recommended								

Appendix A— SWID Extension Schema

The XML Schema below describes the extensions to the SWID specification included in this document. This schema is included in the current draft of this report to assist in review of the schema and related requirements.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
   xmlns="http://csrc.nist.gov/ns/swid/2015-extensions/1.0"
   targetNamespace="http://csrc.nist.gov/ns/swid/2015-extensions/1.0"
   elementFormDefault="qualified" attributeFormDefault="qualified"
   version="1.0-draft4">
   <xs:annotation>
      <xs:documentation>This schema represents extensions to ISO/IEC 19770-2:2015 as
defined by the NISTIR 8060 Final Draft (Draft 4).</xs:documentation>
   </xs:annotation>
   <xs:attribute name="pathSeparator" type="xs:string">
      <xs:annotation>
         <xs:documentation>Used to specify the path separator character used in containing
&lt;File&gt; and &lt;Directory&gt; elements.</xs:documentation>
         <xs:documentation>See guideline GEN-22.</xs:documentation>
      </xs:annotation>
   </xs:attribute>
   <xs:attribute name="envVarPrefix" type="xs:string">
      <xs:annotation>
         <xs:documentation>Used to specify the character(s) used to prefix environment variables
used in containing &lt;File&gt; and &lt;Directory&gt; elements.</xs:documentation>
         <xs:documentation>See guideline GEN-23.</xs:documentation>
      </xs:annotation>
   </xs:attribute>
   <xs:attribute name="envVarSuffix" type="xs:string">
      <xs:annotation>
         <xs:documentation>Used to specify the character(s) used to suffix environment variables
used in containing &lt;File&gt; and &lt;Directory&gt; elements.</xs:documentation>
         <xs:documentation>See guideline GEN-24.</xs:documentation>
      </xs:annotation>
   </xs:attribute>
   <xs:attribute name="mutable" type="xs:boolean" default="false">
      <xs:annotation>
         <xs:documentation>Used to specify that the file referenced by the containing
&lt;File&gt; element is likely to change. Such a change would then modify the file's size and
hash value, causing these values to differ from those defined in the SWID
tag.</xs:documentation>
         <xs:documentation>See guideline PRI-11 and PRI-12.</xs:documentation>
      </xs:annotation>
   </xs:attribute>
```

```
  <xs:simpleType name="PatchEventType">
    <xs:annotation>
      <xs:documentation>This type defines the types of file actions that can be peformed when
applying a patch.</xs:documentation>
    </xs:annotation>
    <xs:restriction base="xs:token">
      <xs:enumeration value="modify">
        <xs:annotation>
          <xs:documentation>Indicates that the patch modifies the pre-existing installed
file.</xs:documentation>
        </xs:annotation>
      </xs:enumeration>
      <xs:enumeration value="remove">
        <xs:annotation>
          <xs:documentation>Indicates that the patch removes the pre-existing installed
file.</xs:documentation>
        </xs:annotation>
      </xs:enumeration>
      <xs:enumeration value="add">
        <xs:annotation>
          <xs:documentation>Indicates that the patch installs a new file that did not
previously exist.</xs:documentation>
        </xs:annotation>
      </xs:enumeration>
    </xs:restriction>
  </xs:simpleType>
  <xs:attribute name="patchEvent" type="PatchEventType">
    <xs:annotation>
      <xs:documentation>Used to identify what operation the patch performs on the file
referenced by the containing &lt;File&gt; element.</xs:documentation>
      <xs:documentation>See guideline PAT-2 and PAT-3.</xs:documentation>
    </xs:annotation>
  </xs:attribute>
</xs:schema>
```

3051

Appendix B—Acronyms

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

Acronym	Definition
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
CMDB	Configuration Management Data Base
CPE	Common Platform Enumeration
DSS	Digital Signature Standard
FIPS	Federal Information Processing Standards
HTML	Hypertext Markup Language
IANA	Internet Assigned Numbers Authority
IDNA	Internationalizing Domain Names in Applications
IEC	International Electrotechnical Commission
IETF	Internet Engineering Task Force
IP	Internet Protocol
ISBN	International Standard Book Number
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
U.S.	United States
US	Usage Scenario
USB	Universal Serial Bus

US-CERT	United States Computer Emergency Readiness Team
W3C	World Wide Web Consortium
WAN	Wide Area Network
WFN	Well-Formed CPE Name
XML	Extensible Markup Language
XPath	XML Path Language
XSD	XML Schema Definition

3054

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].

[NISTIR 8085] Cheikes, B. A., and Waltermire, D., *Forming Common Platform Enumeration (CPE) Names from Software Identification (SWID) Tags*. National Institute of Standards and Technology Interagency Report 8085, December 2015. http://csrc.nist.gov/publications/PubsNISTIRs.html#NISTIR8085 [accessed 12/17/15].

[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].

[W3C-langtags] Ishida, R. *Language tags in HTML and XML*. W3C Article, September 2009. http://www.w3.org/International/articles/language-tags [accessed 12/1/2015].

[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].

3056

Appendix D—Change Log

Due to subsequent changes in updated drafts, section and requirement numbers in this changelog might not be accurate to the current draft revision of the document.

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 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.

Release 4 – December 15, 2015 (Fourth public comment draft)

Functional Additions/Changes/Removals:

- Many revisions throughout Section 1
  - Updated Section 1.1 to more clearly present the purpose and benefits of SWID tags.
  - Revised Section 1.3 to include an additional (fourth) audience - *providers of software build, packaging, and installation tools*.
  - Updated Section 1.4 include the new audience and make clear that the report is aligned with the recently released ISO/IEC 19770-2:2015 specification.
- A new Section 2 has been inserted to review key SWID tag concepts that are helpful for understanding the different types of tags, how tags are created, and how tags are made available for use.
  - Section 2.1 was updated including a revised graphic and associated text to clarify interoperability during various life cycle stages. Additional changes in 2.1 include:
    - Section 2.1.2 adds clarification regarding handling of existing tags when a software product is updated
    - Section 2.1.2 provides an updated Figure 2 (Primary Tag Relationships) and associated text reflecting SWID tag associations among multiple software products (e.g., relationship of a software component to a parent software or suite of products).
    - Section 2.1.3 is expanded to further explain how patch SWID tags will include information to document a relationship to the primary tag of a patched product.
  - A new section 2.3 has been added to describe various factors regarding the placement of SWID tags.
  - A summary for the new Section 2 was added in 2.4.
- The new Section 3 (formerly part of Section 2) presents a high-level description of SWID tag data elements as specified in the SWID specification.
  - Section 3.1 provides additional information regarding the <SoftwareIdentity> element, particularly section 3.1.2 that was updated for more detailed entity usage. Section 3.1.3 adds guidance regarding how to record evidence collected from a shared location (e.g., a NAS device).
  - Section 3.2 clarifies the application and use of an XML digital signature to a tag to address cybersecurity risks.
- Section 4 (formerly Section 3) has been updated throughout.
  - Section 4.2 expands general guidance regarding tag creation authority (e.g., Authoritative and Non-Authoritative Tag Creators).
  - Section 4.3 largely expands on the use of <SoftwareIdentity> elements. It broadens support for SWID for an international audience, recognizing the need to permit tag creators to provide language-dependent attribute values in region-specific human languages. Three new general guidelines (GEN-2, GEN-3 and GEN-4) have been added regarding language and <Entity> elements
  - Section 4.3 also recognizes that non-authoritative tag creators may be less able to provide reliable @regid information. This understanding leads to the two additional new guidelines (GEN-5 and GEN-6) related to registration identifiers.

- Section 4.4 was expanded to support provision of more detailed information about entities. The section furnishes information to help prevent unnecessarily complex entity specifications, distinguish between authoritative and non-authoritative tags, and furnish additional information about the software creator. To that end, the text includes a new guideline (GEN-7).
- Section 4.5 was expanded to provide guidance regarding implementing <Link> elements. The section provides guidelines for establishing relationships of various kinds between tags and other documents, including other tags. The guidelines specifically address two issues: linking a source tag to a known target tag and to a collection of tags. The text describes three new guidelines (GEN-11, GEN-12 and GEN-13) related to these relationships. Four detailed examples are provided to assist users in applying the linking guidance.
- Section 4.6 provides expanded information regarding the use of <Payload> and <Evidence> elements to provide sufficient file information for asset management and cybersecurity needs.
- A new Section 4.6.3 provides detailed guidance regarding the use of path separators and environment variables to reference <File> and <Directory> elements. The section includes several new guidelines (GEN-22, GEN-23 and GEN-24) and three new extension attributes (@pathSeparator, @envVarPrefix and @envVarSuffix).
- In support of the aforementioned internationalization need, a new Section 4.7 has been added to describe providing attribute values in multiple languages. Because the SWID specification permits the @xml:lang attribute be used on any tag element, this enables tag creators to implement multilingual tags, tags which provide language-dependent attributes in more than one language.

- Previous guidance regarding implementation of digital signatures has been removed. While application of an XML digital signature to SWID tags is addressed in the ISO specification and helps address cybersecurity risk, the authors did not find sufficient consensus in the community to recommend specific additional guidelines regarding this topic. As use of digital signatures in SWID tags increases, additional guidelines may be needed.
- Several previous guidelines (GEN-14 through GEN-17) regarding tag identifier references have been removed since these generally were covered by other guidance.
- Section 5 received extensive revision regarding implementation guidelines that are specific for implementation guidelines that are specific for each of the four tag types defined in Section 2.1: corpus tags, primary tags, patch tags, and supplemental tags.
  - In Section5.1, several guidelines were removed regarding the use of well-known public lists of version schemes. Such a resource does not exist at this time. Thus, guidelines COR-4 and COR-5 from Draft 3 have been removed as well.
  - In Section 5.2, the former PRI-6 and PRI-7 version scheme guidelines were removed for the reasons described above.
  - Section 5.2.3 has been updated to clarify some of the challenges and benefits regarding inclusion of file information in <Payload> and <Evidence> elements. In the interest of minimizing the possibility of false positives from frequent file changes, the document provides two new guidelines (PRI-11, PRI-12) for tag

creators to explicitly mark all mutable files listed in a tag's payload element with a special value, an extension attribute "mutable".
    - Due to the new, extended guidance regarding linking tags, the former subsection about "Linking Patch Tags to Related Tags" was removed, including removal of the former PAT-2, PAT-3 and PAT-4 guidelines.
    - Section 5.4.2 provides additional information regarding linking supplemental tags, in connection with the earlier guidance in Section 2, and points out that a supplemental tag may supplement any type of tag, including other supplemental tags.
- In Section 6, usage scenarios were updated including a full refresh of all cross-references to the former sections and guidelines.
    - Section 6.4 was updated to add guideline groups for Internationalization, Tag Entities and Tag Linking. Groupings were removed for XML Signature, Tag Identification and Patch Integrity Verification, since the related guidelines have also been removed.
- The former Appendix A (Forming Common Platform Enumeration Names) has been removed from the document and is now released as a separate NIST Internal Report 8085, Forming Common Platform Enumeration (CPE) Names from Software Identification (SWID) Tags.
- A new Appendix A has been added to provide a schema for the new extended attributes defined in this report.

Editorial Changes:

- Made minor editorial revisions throughout the report.
- Divided the original Section 2 into two separate sections.
- Updated the acronym list (Appendix B) and the references (Appendix C) to take into account related changes made elsewhere in the report.