Standardizing SBOM within the SW Development Tooling Ecosystem

Many end-user organizations across the world are facing operational and supply chain related questions about whether the software they are using to do their day-to-day work and support their ongoing business areas is authentic and unaltered, contains known vulnerabilities, or whether their use is proper and legal given the licensing terms placed on the constituent parts of that software by its developers. The fact that software is being used to run more and more of the critical aspects of each of our organization's business, embedded, and cyber-physical systems makes it these pressing and unavoidable questions for almost everyone.

Much attention has focused on identifying the needs for Software Bill of Materials (SBOM) information in end-user organizations, which extends to understanding the software content of their operational systems, the supplier communities of that software, whether that equipment has software embedded or they are directly supplying software. At the same time the software development tools ecosystem organizations, those who will be key in supplying the tools that are foundational to supplying automated SBOM information, need to be engaged and the SBOM standardization needs to support and help the integration of the different development, assessment, and analysis tools into the emerging DevSecOps abilities being explored across the market.

Minimal Common Information Needs

The following discussion explores several different ways an SBOM can be used to help understand and address software supply chain and operations risks in end-user organizations and identifies the minimal information needed in a SBOM to support those uses[1]. Following that is a discussion on the role that this same data could have in tool-to-tool exchanges within the software tools ecosystem and ideas about how that data could be captured as a specific tool-to-tool information exchange standard.

To be useful to end-user organizations an SBOM needs to include the basic information that enables correlating and connecting related information about the software as it moves through the supply chain and into operations. It is also critical that the SBOM be as small and concise as possible to help ease the integration and adoption and keep management of SBOM data as simple as possible. A key idea in driving that minimalization is to aim at having the data in the SBOM static and fixed. Any information that will change and evolve over time should be separately managed so it can be correlated and linked to the SBOM, but it should not be included in the SBOM. That does not mean that an SBOM is static. Better information or more refined information about a specific version of software may be created, discovered, or recovered and an updated SBOM authored with the new/revised information could be provided.

For example, information about what components contain components is something key to understanding the nature of the software linked to an SBOM. Once the component is created this information does not change and provides the organizational context for understanding what other software components were included by the developers of the components and having it in the SBOM seems essential to the functionality of SBOMs in an enterprise.

However, information about vulnerabilities in the software should be separate from the SBOM since new ones are being discovered all the time and the knowledge about what to do about the vulnerability, how to mitigate it and address it also changes. To be operationally useful any organization planning to utilize SBOMs for security of their systems will need to have activities that can connect the information in the SBOMs to their vulnerability management activities, but that mapping is a separate activity, one enabled and enhanced by future pervasive existence of SBOMs for all of the software in the enterprise.

On the other hand, an example of one type of information that could be considered for inclusion in the SBOM is the licensing terms for the software when it is created. Organizations can change licensing terms whenever they like but for a specific version of the software that licensing is fixed at the point of release/creation and can safely be included in the SBOM. If the licensing terms of software are of concern the organization would need an organizational ability to understand and map the terms of licensing of the specific SBOMs for the different versions of software they will have in their enterprise and use they will

[1] Based primarily on the National Telecommunications and Information Agency's (NTIA's) Software Transparency efforts <www.ntia.doc.gov/SoftwareTransparency>

need to analyze and determine the relevance of any changes to those licenses from one version of the software to another, but the germane licensing information would be available in the SBOMs for the different versions of the software.

Finally, for those with higher assurance concerns, information about the creator of the software, the source location of the components that are included in the software, and the compilation details, options, and tools used to build the software, would be a extremely valuable type of information to include in the SBOM since this information is static once the software is created, and by moving through the supply chain along with the software, that information would be readily available to downstream consumers of that software. This type of assurance information is one that would be very difficult, if not impossible, to obtain otherwise.

Usage Scenarios for end-user use of SBOMs

Usage Scenario 1: helping parties unambiguously **refer to, transfer, or purchase** a specific software component would require the ability to articulate the component author, component name, and the version of the software as well as other components that may be utilized by that component, or communicating that there are no other components being utilized. If that cannot be stated, then the unknown state of utilization of other components needs to be communicated. The author of the SBOM should also be conveyed, allowing it to be different than the component author, as well as the time/date of the SBOM itself. Consideration for denoting the basis of the information captured is being discussed. For example, created from either: a) build tool; b) SCA type tools; c) parsing a package manager; or d) manual. Depending on the specific part of the SBOM information, some approached may be more "accurate" or "repeatable" than others but having the information will allow the recipient of the SBOM to make their own assessment in a more straight-forward manner. Finally, a way to bind that information to the actual software and provide a unique binding to it through either a hash of the software, a GUID, or a UUID, must be included in the SBOM.

Usage Scenario 2: helping determine whether appropriate choices were made for securing the software during the creation process would require, in addition to the basic unambiguous reference information from Usage Scenario 1, information about the compilation and formulation options used in transforming the source components and parts into the resultant software. For example, were address space layout randomization (ASLR), data execution prevention (DEP), solutions that monitor for stack overflows, or that invalidate writes to adjacent memory in stacks invoked? Another example would be when dynamic linking is used, what is the intended operating system environment? This information will allow that those dynamic libraries in the SBOM be the specific ones for that environment versus all versions. Finally, was reproduceable compilation used? When done for each software component this is known as **pedigree**.

Usage Scenario 3: helping determine whether the software is authentic would require, in addition to the basic unambiguous reference information from Usage Scenario 1, capturing information about the organizations that handled or were involved in the creation or the sourcing of the software and its constituent parts. This is often referred to as **provenance** and focuses on establishing the chain-of-custody of the software and is traditionally accomplished by using signing techniques to validate the source of each item as it moves along the supply chain.

Usage Scenario 4: helping determine whether the software and SBOM information are unaltered would require, in addition to the basic unambiguous reference information from Usage Scenario 1, information about the **integrity** of that software, its constituent parts, and the SBOM itself, as they pass along the supply chain. This is often accomplished by using a hash of the software components in the SBOM and using signing techniques to ensure the integrity of the SBOM as it passes along each link of the supply chain from the creators of the software to the users of it.

Usage Scenario 5: helping determine whether the planned use of the software item is aligned with its **intellectual property constraints** would require the basic unambiguous reference information from Usage Scenario 1 and the license(s) for the software and its components. This is often attempted by mapping the information available in public open source repositories about current license terms for the software components but having the information for a specific piece of software included in the SBOM would

eliminate uncertainty and limit the additional effort needed to obtain authoritative information upon which to make decisions.

Usage Scenario 6: helping determine whether there are any **known software vulnerabilities** in a software item or any of the constituent parts would require the basic unambiguous reference information from Usage Scenario 1. Determining whether there are known vulnerabilities is normally accomplished by mapping the up-to-date information available in publicly known vulnerability repositories about known vulnerabilities to the software components in a software item. By encouraging indexing public repositories with SBOM identities this mapping will be more effective and efficient, and possibly automatable. Capturing and conveying information about the non-impact of vulnerabilities in incorporated components is of strong interest and could be accommodated by including a notes/comment field where statements about such analysis could be conveyed.

Usage Scenario 7: helping determine whether the software and its components are **secure, safe, and resilient** would require the basic unambiguous reference information from Usage Scenario 1 but would be correlated to that information, not included in the SBOM itself. This is currently accomplished either by the developing organization or another, providing the results of static, dynamic, threat and other evaluations and analysis and providing an assurance case representing how the claims of safety, security, and resilience are supported by evidence, thus providing evidence-based **assurance** about the software, tied to the SBOM for the software.

Usage Scenario 8: helping determine what the **software providing a service** is at the point of execution would require the basic unambiguous reference information from Usage Scenario 1. This could be accomplished by the service providing the SBOM Usage Scenario 1 information for archival logging at the point that software service is invoked so it would be available for retrospective analysis about vulnerabilities found to have been in the software that was used by the service. Having this information would allow further analysis of whether the vulnerability in the service was exploited when the organization used it. This does not replace the need to investigate the full functionality of the service being invoked and having confidence that its functionality does not include harmful capabilities. If Usage Scenarios 2, 3, 4, and 5 are of interest to an organization, the fields needed to provide pedigree, provenance, integrity, and intellectual property constraints would need to be logged.

Usage Scenario 9. Helping determine whether a required/desired sequence of steps for the software and its components has been completed in a specified order, and with no additional steps would require, in addition to the basic unambiguous reference information from Usage Scenario 1, and the provenance information from Usage Scenario 4, a list of ordered steps, and requirements for each step. This information is referred to as **supply chain sequence integrity** and provides assurance to the downstream consumers that software has completed expected/required steps, and that no unexpected and potentially malicious steps have been inserted into the documented supply chain sequence. This could be of great utility where validating whether the specified tool chain of a DevSecOps environment were followed.

Enabling Tool-to-Tool SBOM Exchanges

To enable tool-to-tool use within the iterative software development/integration and testing ecosystem of tools, a very small SBOM standard format/structure that the software tool ecosystem can utilize as they move code and components through their workflows from partner tool to partner tool is prudent and essential.

When someone at a development shop has a need to externalize the SBOM information, the SPDX file format[2] and SWID structure[3] may be capable and appropriate for that exportation, but in discussions with many of the software development tool creators they don't seem to see either SWID or SPDX as something

[2] The Software Package Data Exchange (SPDX®) specification <spdx.github.io/spdx-spec/>
[3] SWID Tag: "ISO/IEC 19770-2:2015 establishes specifications for tagging software to optimize its identification and management" or NIST Internal Report (NISTIR) 8060:Guidelines for the Creation of Interoperable Software Identification (SWID) Tags <doi.org/10.6028/NIST.IR.8060>

they would utilize internally within their community when interacting with other tools.

The native, internal use of a standard SBOM is key to getting consistent and broad uptake in the market and moving of this data to the places it is needed. This is especially true of the software provenance (i.e., chain of custody) information and the pedigree (i.e., formulation/compilation choices) information since it is extremely hard to recover or extract that information accurately and consistently after the fact.

The Software Development Tooling Ecosystem

Historically software was created from scratch, starting with an empty file and then creating the overall design and flow of the needed functionality. Typical examples included specific nuances of handling interactions with the user (user interface), storing and retrieving information, and communication with other applications and services, as well as all of the general "housekeeping" activities entailed with well-behaved and operationally efficient software that performs well in-use. Software was compiled into a binary form that would run on the target system, utilizing standard interfaces with the operating system through standard libraries and functions that were linked at the time of compilation. Figure 1 depicts the typical components of a software compilation environment.

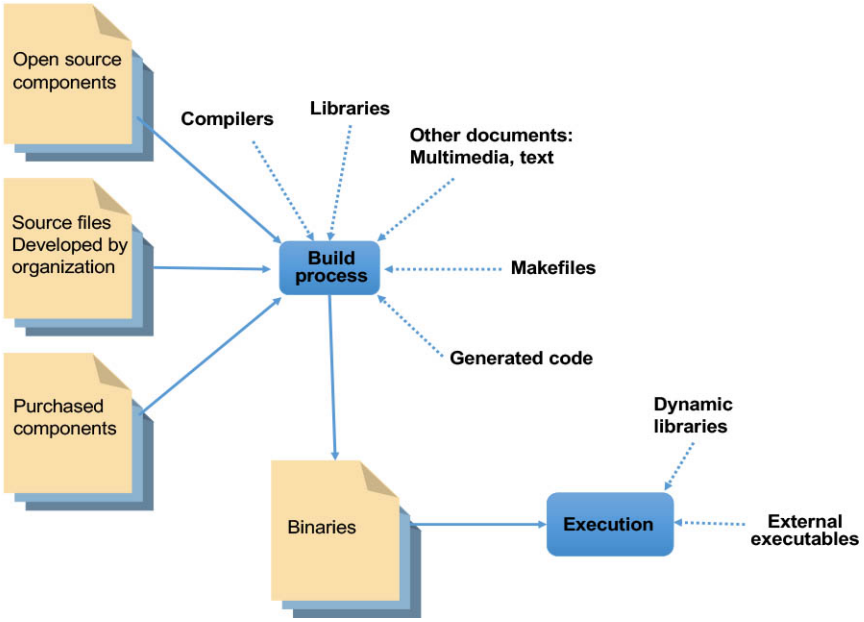

Figure 1: Software compilation components

In this type of development environment, the compiler is the main tool, along with the code editor that provides the mechanisms for creating and modifying the source files. These main components in turn work with the code repositories that open source and purchased components come from. Integrated Development Environments (IDEs) are suites of more elaborate linked code editors that are often used as the development tool of choice in large enterprises and focused coding groups.

The contemporary approach to building much of today's software has evolved towards more of an assembly and refinement approach, where the development starts with an "off-the-shelf" framework that has basic input/output, screen handling, house-keeping, and interactions already in place. The interactions with a host operating system are already implemented, and the developer can immediately work to see what supporting functional components are needed for the specific application functionality they are creating. Additionally, the applications can be tailored for different environments, such as embedded, web, cloud, mobile, or others with appropriate orchestration of platform specific versions and features brought together as part of the build capabilities.

With this style of development, the ability to find licensing information and vulnerability information for the third-party components and frameworks can become a critical element of an organization's build operations.

The unfortunate truth is that most third-party components use other third-party components, so finding out what those components are and what licensing or patching issues are related to them has spawned a collection of techniques and capabilities for recovering this type of information about the software composition.

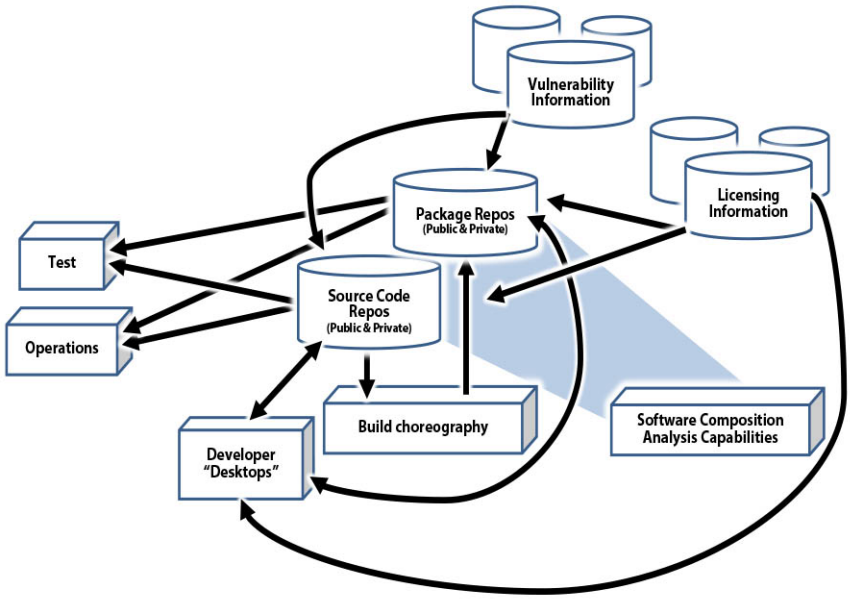

Figure 2: Software Development Tooling Ecosystem

Finally, the software that is created or modified will need to be tested and hopefully put into operations. Figure 2 illustrates the numerous paths that software components (or information about software components) can be expected to flow between, throughout the different parts of the software development tooling ecosystem.

The Software Development Tooling Ecosystem Member Organizations

Within the various grouping of capabilities depicted in Figure 2 there are many products (both open source and closed source) that would need to incorporate native SBOM related abilities to create, manipulate, manage, test, and control the software and its SBOM. For the foreseeable future, a large but hopefully diminishing amount of the software components available for reuse by third parties will not have SBOMs created by the original authoring organization and so the Software Composition Analysis community, identified below, will be often relied upon to create and populate SBOMs for these components. The objective of this effort is to have one structure that both communities can leverage and provide SBOMs where desired. The following six tables provide listings of the capabilities available today to support development and composition analysis efforts.

The 6 tables illustrate some of the current members of: 1) IDEs; 2) Frameworks; 3) Cloud Tools; 4) Source Code & Package Repositories; 5) Build Choreography Capabilities; and 6) Software Composition Analysis Capabilities. Additional tables for testing tools and operational management tools that could utilize and leverage SBOMs will be created as that data is gathered.

Table 1: IDEs

Android Studio	Code Blocks	GoLand	MPLAB	Spiralogics
AppCode	CodeCharge Studio	IDLE	NetBeans	Application
Atom	CodeLobster	IntelliJ IDEA	PhpStorm	Architecture
BlueJ	CodePen	LINX	Pycharm	WebStorm
CLion	DataGrip	Microsoft Visual	Rider	Xcode
Cloud9 IDE	Eclipse	Studio	RubyMine	Zend Studio

Table 2: Frameworks/Libraries/Tools

.NET	Cordova	Hadoop	React.js	Visual Online
Angular	CryEngine	HTML5 Builder	Ruby on Rails	Vue.js
Ansible	Django	Laravel	Spring	Xamari
Apache Spark	Drupal	Node.js	TensorFlow	
ASP.NET	Express	Pandas	Torch/PyTorch	
Bootstrap	Flask	Puppet	Unity D	
Chef	Flutter	React Native	Unreal Engine	

Table 3: Cloud Tools

Azure	Cloud Foundry	Kwatee	Red Hat
AWS CodeBuild	Google Cloud Build	Pivotal	

Table 4: Source Code & Package Repositories

Amazon ECR	Codebase	Google Container	Launchpad	Savannah
Assembla	Docker	Registry	Maven	SourceForge
Azure Container	GitHub	JFrog Artifactory	Nexus (Sonatype)	SourceRepo
Registry	GitLab	JFrog Xray	Phabricator	Subversion
Beanstalk	Glitch	inedo	ProjectLocker	Unfuddle
Bitbucket		Kubernetes	Repository Hosting	

Table 5: Build & Build Choreography Capabilities

Ansible	Buildroot	GCC	Strider CD	Vagrant
Autorabit	CircleCI	Gitlab CI	TeamCity	
Bamboo	CMake	GoCD	Terraform	
Bitrise	CruiseControl	Integrity	Travis CI	
Buildkite	Final builder	Jenkins	Urbancode	

Table 6: Software Composition Analysis Capabilities

Black Duck Software Composition Analysis (Synopsys)	FlexNet Code Insite (Flexera)	Sonatype
CAST Highlight (CAST Software)	Ion Channel	Snyk
Finate State	Insignary	WhiteSource
	SourceClear	

Potential SBOM Elements to support the Described Usage Scenarios

Within the nine Usage Scenarios discussed on pages 2 and 3 of this paper there were several elements of information described as the scenarios were elaborated. The following will attempt to capture the minimal elements to support the Usages described. The potential SBOM elements, shown in the middle of Figure 3 will be mapped to the Usages they are needed for and current thoughts on external information that would be correlated to the SBOM will be shown in the right-side of the subsequent Figures 4 through 12.

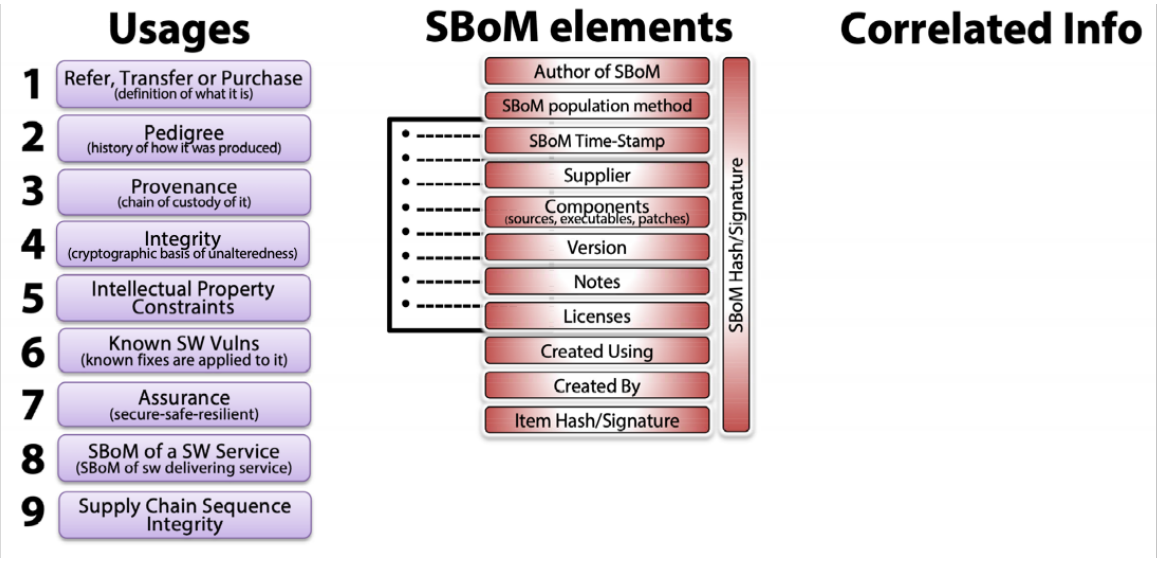

Figure 3: Software Bill of Materials Potential Elements for Discussion

Usage Scenario 1

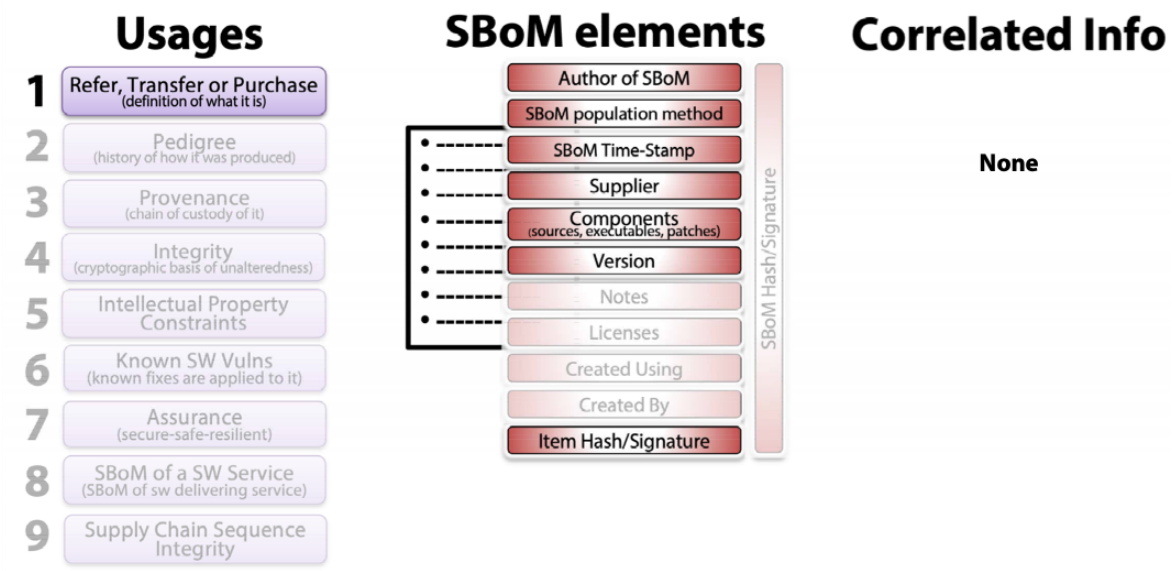

Figure 4: Software Bill of Materials Potential Elements for Refer, Transfer, or Purchase

Usage Scenario 2

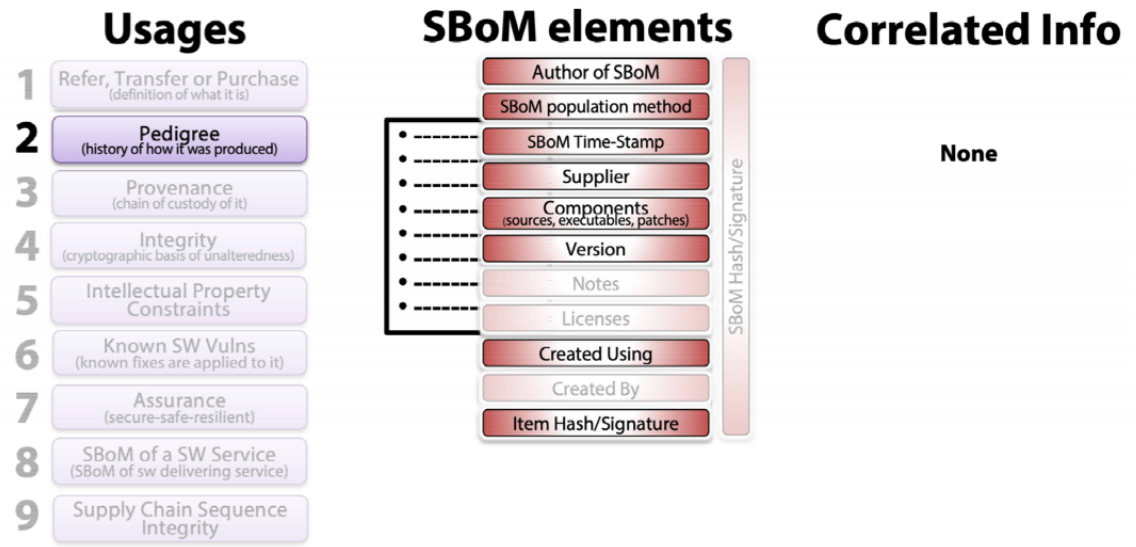

Figure 5: Software Bill of Materials Potential Elements for Pedigree

Usage Scenario 3

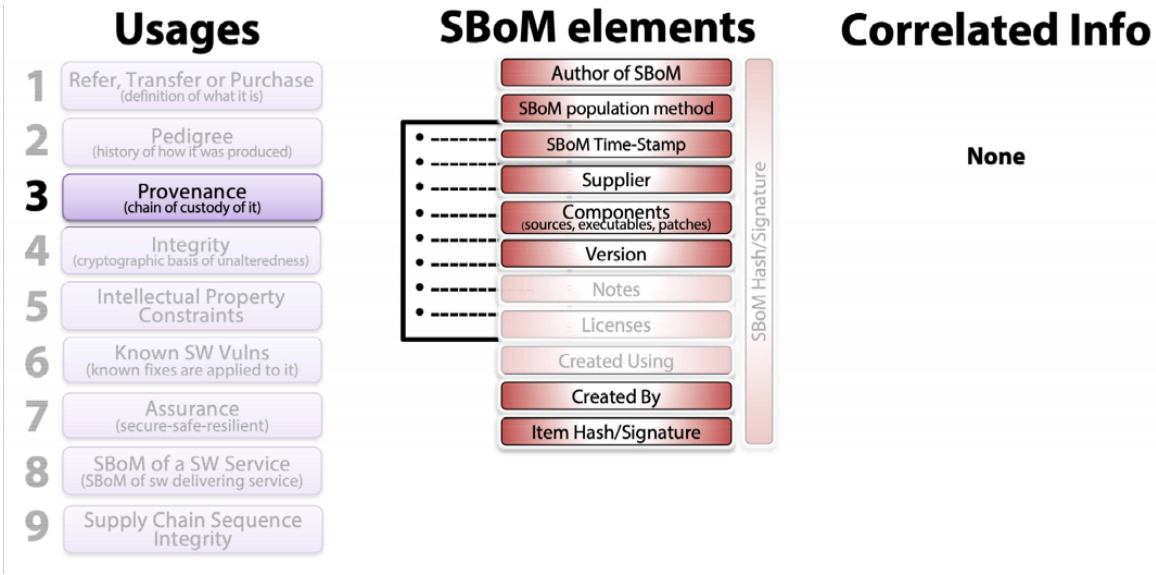

Figure 6: Software Bill of Materials Potential Elements for Provenance

Usage Scenario 4

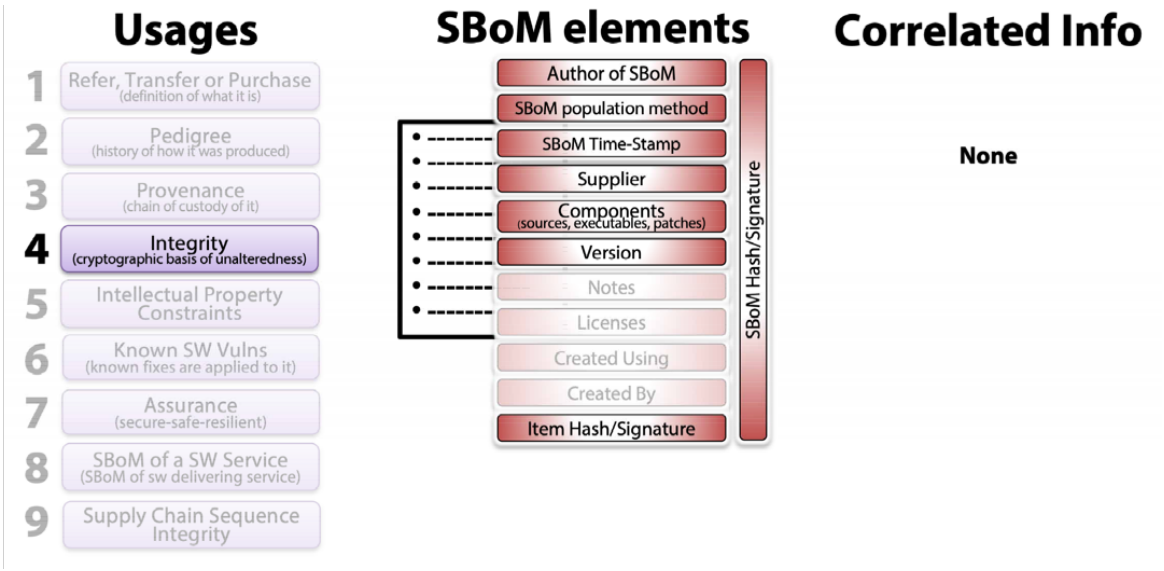

Figure 7: Software Bill of Materials Potential Elements for Integrity

Usage Scenario 5

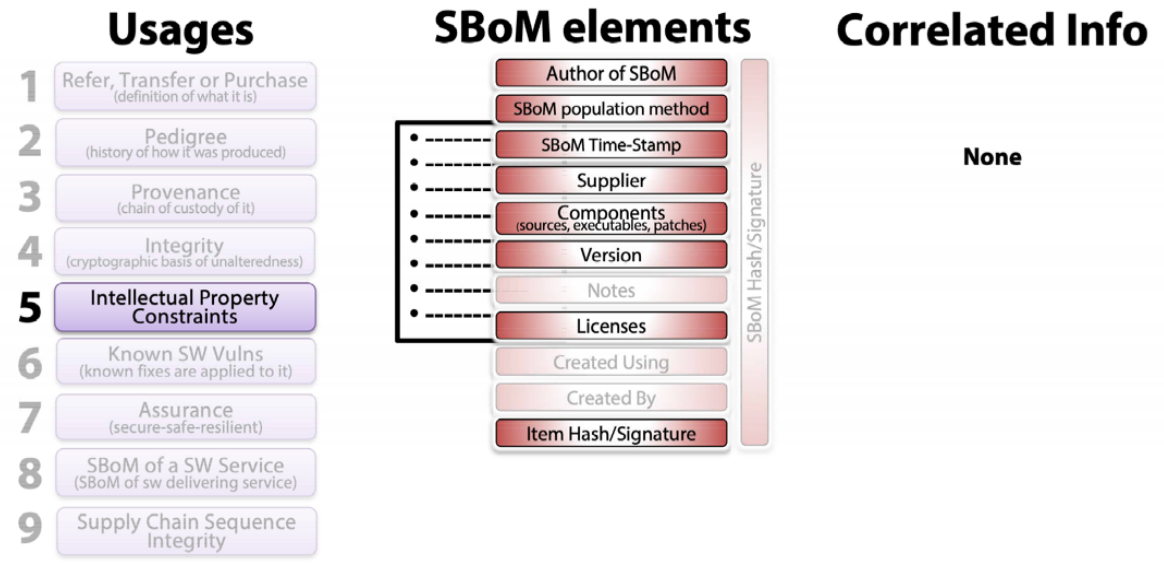

Figure 8: Software Bill of Materials Potential Elements for Intellectual Property Constraints

Usage Scenario 6

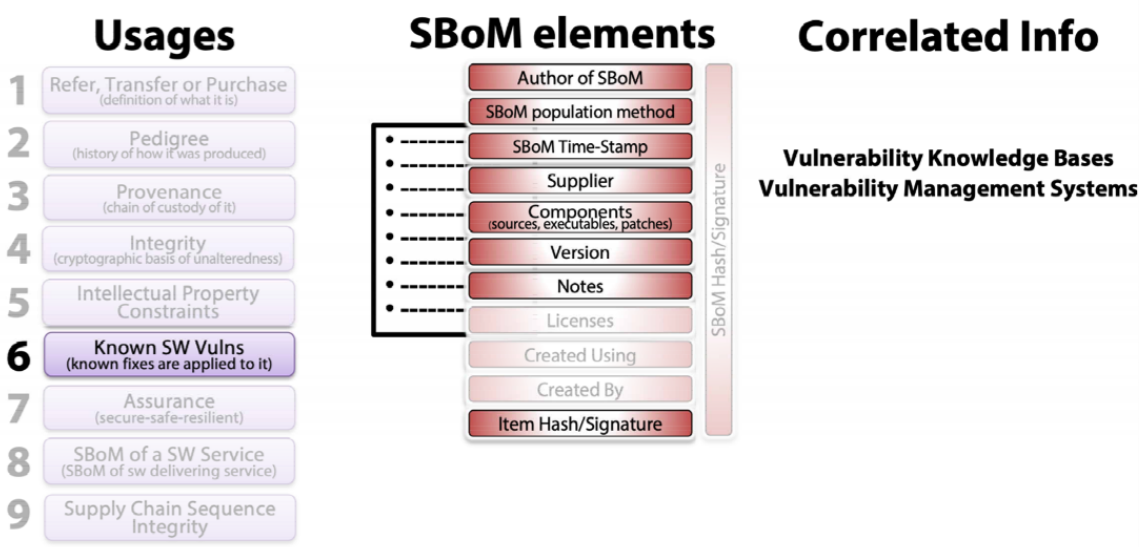

Figure 9: Software Bill of Materials Potential Elements for Known Software Vulnerabilities

Usage Scenario 7

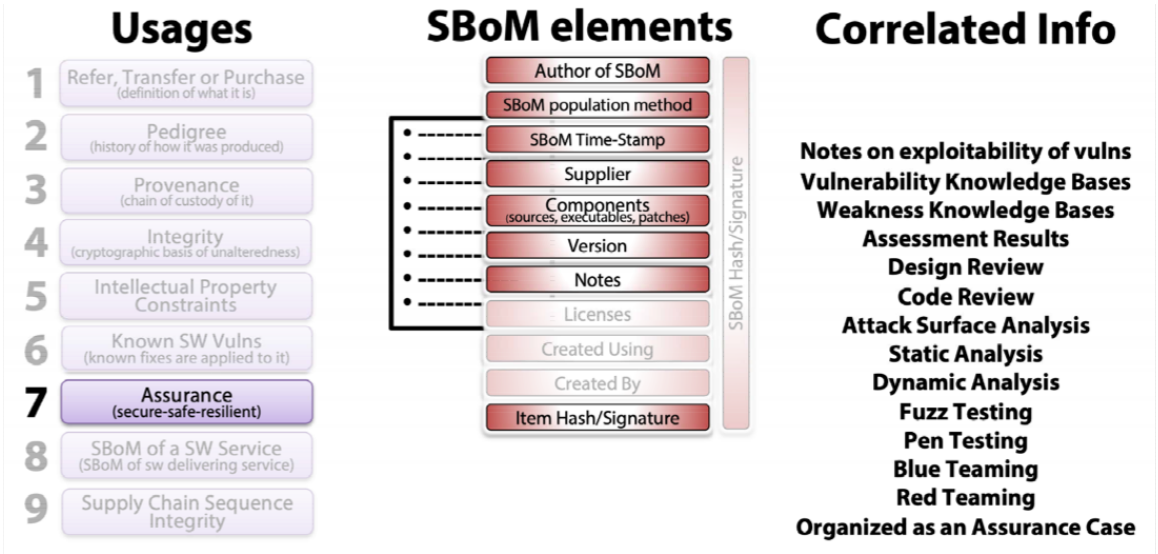

Figure 10: Software Bill of Materials Potential Elements for of Assurance

Usage Scenario 8

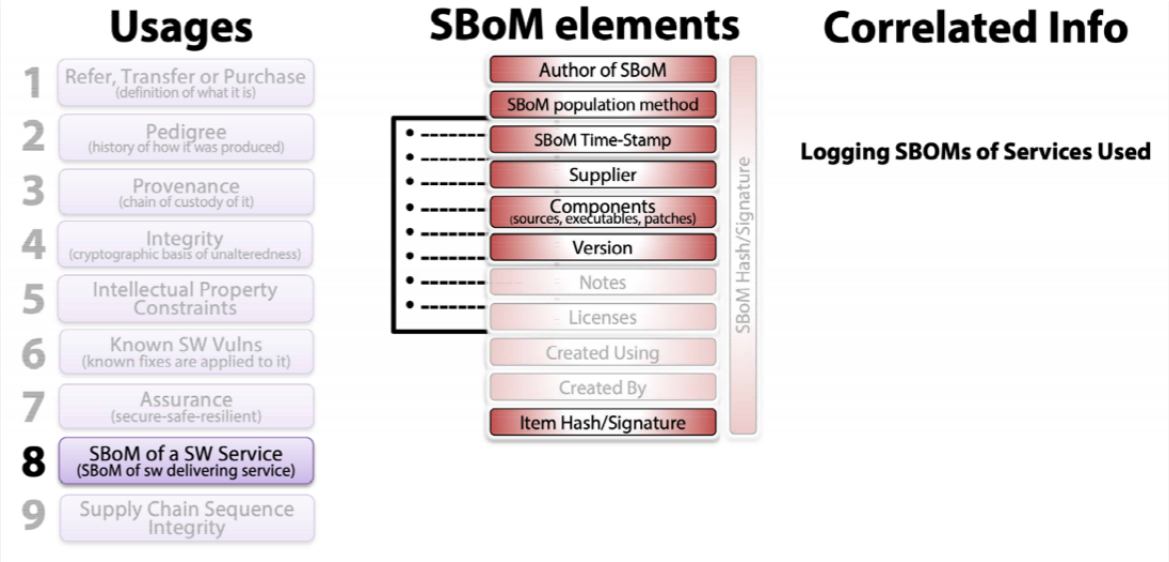

Figure 11: Software Bill of Materials Potential Elements for SBOM of a Software Service

Usage Scenario 9

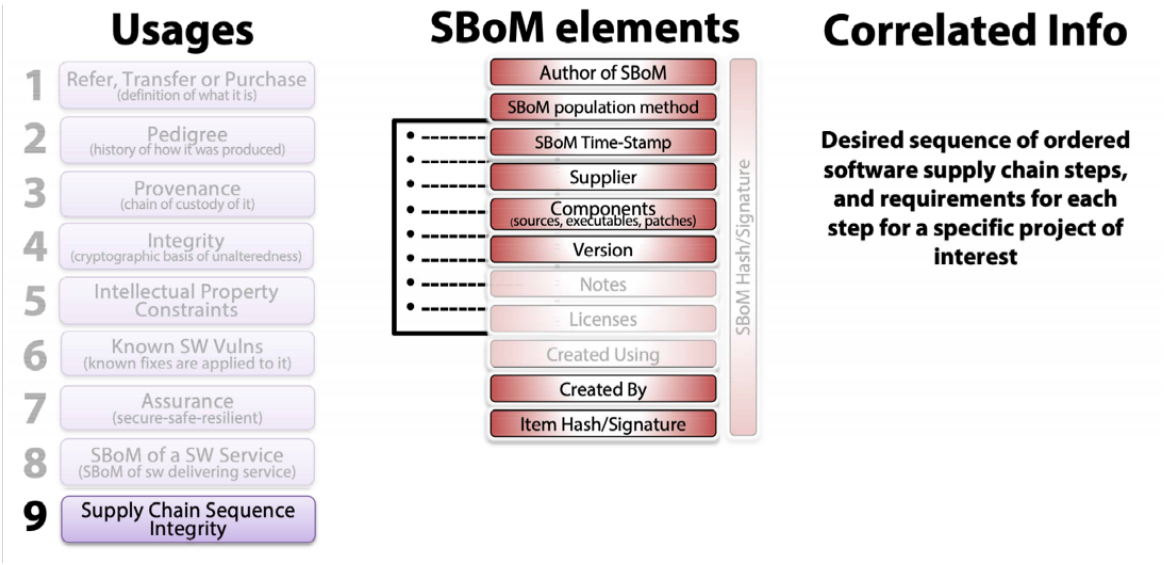

Figure 12: Software Bill of Materials Potential Elements for Supply Chain Sequence Integrity