Software Suppliers Playbook: SBOM Production and Provision

Version 1.0
2021-11-17

NTIA Formats and Tooling Working Group

Overview

This playbook outlines workflows for the production of Software Bills of Materials (SBOM) and their provision by software suppliers. "Supplier" is broadly defined to include software vendors supplying a commercial product, contract software developers supplying a software deliverable to clients, and open source software (OSS) development projects making their capabilities publicly available.

Summary of Workflow

The diversity of organizations that create software, and the need for SBOM creation across a range of software and systems, means that organizations will produce SBOMs with a wide range of tools and processes. The U.S. government has published a list of elements, including data fields, automation capability, and operational considerations, that it expects to be included in an SBOM regardless of how it is created.[1] Much of this document helps characterize the considerations and decisions that teams will make, based on their existing tools and processes and their levels of technical maturity.

We can roughly characterize the SBOM production process in the following steps:

1. Identify software components included in a software deliverable.

2. Acquire data about components used in a software deliverable.

3. Import component data into a structured SBOM format.

4. Validate SBOM to ensure that format is valid, and the baseline attributes are present.

Relevant Workflow Differentiators

Every organization will have distinct needs and capabilities for software creation and distribution: an existing set of tools and processes, as well as ongoing migrations to different tools and processes as part of engineering modernization efforts. For this reason, there will be variations in workflows for generating SBOMs between organizations or even within the same organization.

Below, we characterize some of those variations, and how different conditions will guide organizations toward the efficient and effective creation of SBOMs.

Current Best-Practice vs. Non-Automated Engineering Processes

The generation of the source-level SBOM ("pre-build" SBOM) can be part of a version control system or can be created by tools that mine the inputs to the product-build pipeline. Effective hashing of the source files (and hashes of any binaries) that go into a build pipeline is useful for ensuring the identification of key components. This is most important when looking for vulnerabilities before a product has been created. Some high-assurance and safety-critical use cases may require traceability from the build to the source-file level; having an SBOM of the source code, associated data, and artifacts incorporated in the build may be required. However,

[1] The Minimum Elements For a Software Bill of Materials (SBOM) https://www.ntia.gov/files/ntia/publications/sbom_minimum_elements_report.pdf (2021)

pre-build SBOMs may represent vulnerable source code which is not included in the final executable.

Automated Workflows: Production of SBOMs as a Build Artifact
Current best-practice engineering workflows, such as git for software version control and Continuous Integration/Continuous Delivery (CI/CD) pipelines, enable automated collation and generation of SBOMs from a pipeline build ("build-time" SBOM generated during compilation of the source code or the software packaging process). Because these automated processes derive data necessary to build software, the resulting SBOMs are generally free of human manual entry errors and contain more authoritative software component identities. Automated SBOM generation can also automate the signing of SBOMs, which provides additional auditability for both the supplier and downstream consumers.

SBOMs can be created by leveraging tools that are integrated with build systems, package managers, and CI servers. The creation of SBOMs during a build pipeline has numerous technical and business advantages that are beyond the scope of this playbook. Build-time creation of SBOMs typically involves invoking a tool that works natively with the build system being used. The tool then generates component inventory and all associated metadata that can then be augmented and enhanced throughout the remainder of the build pipeline. The resulting SBOM can be delivered as an additional artifact, specific to the version of the software that was built. Additional auditability is enabled by cryptographic signatures of SBOMs at build time.

SBOM suppliers must decide on an SBOM format that will be generated during their build process. Each of the SBOM formats below contains a mapping from the "baseline" attributes, identified in the Framing document, to the SBOM-specific data elements that have been identified to represent these common elements.

- Software Package Data Exchange (SPDX): https://github.com/lfscanning has SBOMs for some common open source projects.
- CycloneDX: Some examples are here: https://github.com/CycloneDX/sbom-examples.
- Software Identification (SWID): References to SWID examples are included in https://csrc.nist.gov/publications/detail/nistir/8060/final.

If an audit of software licenses is desired, there are source and binary code scanning utilities that can summarize licensing details and generate an SBOM that includes such information. The ability to automate generation of license data within an SBOM can satisfy conditions of open source licenses that require disclosure of incorporated components.

SBOM Production in Build Pipelines and Software Factories
Many organizations automate the creation of SBOMs in their development lifecycles. This is often accomplished in CI/CD pipelines where every build produces an SBOM as an artifact. Most development ecosystems have an optional method for creating SBOMs through the use of build plug-ins. These plug-ins integrate with the underlying build and dependency management

systems to produce one of the supported SBOM formats. This approach is simple to adopt, but may require additional resources to integrate with all build pipelines.

Without consistent and universal SBOM support across all development ecosystems, some organizations have adopted a strategy that helps them accelerate the creation of SBOMs across their organization. Rather than integrating with each build, the CI/CD software factories themselves may be extended to support automated SBOM creation. Pipeline integrations and GitHub actions that perform automatic SBOM creation are examples of this functionality.

Containers Export of SBOMs from a Containerization Process

Software may be delivered as an application or executable. Increasingly, software is delivered in container images, such as Docker/Open Container Initiative (OCI) formats. Container images should have an accurate and complete inventory of components that are derived from all layers of a container image. Each layer may contain various software applications, including the operating system (OS), OS packages, applications and their associated libraries, along with other artifacts that may have been incorporated into various layers. SBOMs that describe container images should aggregate and identify all software from all layers.

Multiple container images may be used together to form an application. How this is performed is beyond the scope of this document. However, the dependency relationships that exist between containers cannot be ignored. Docker Compose and Kubernetes Helm charts are two popular methods of describing containerized resources. Since both describe one or more container images, the aggregation of all software from all images these formats describe should be captured in SBOMs, either as individual SBOMs, or in aggregate. If containers provide services to other containers, these relationships should be captured in the SBOM dependency graph. Refer to External Services: Crossing Trust Boundaries for more information.

SBOMs generated at build time should include the time of build as part of the SBOM.

Customers may require a signature of an SBOM as an element of assurance. To satisfy these customer requirements, SBOMs generated as part of a build process should be signed as part of the build process.

Post-Build SBOMs

Not all delivered software can be part of build-time SBOM generation. Many software capabilities, especially capabilities used in older systems, were not originally developed using current methods for software version control or continuous integration. For these systems, supplier acquisition of the data to populate a “post-build” SBOM should focus on acquiring component data as close to the engineering process as possible, and on explicit acknowledgement of “known unknowns,” which may include globally unique external identifiers. A post-build SBOM may be a collection of SBOMs from various suppliers, processes, and tools. Binary scanning, often performed using code analysis tools, may be used to populate SBOM information concerning components received from upstream suppliers, where such SBOM information would not otherwise be available. Although the code analysis tools may produce SBOMs, these tools may be limited in the ability to identify commercial components.

For non-automated systems and processes, it is important to understand the source of the information for components that are being listed in an SBOM, and how that information was obtained for the SBOM. Salient questions include:

1) Point of Origin: Is there a way to verify that components enumerated in an SBOM came into the supplier organization from a specific point of origin (i.e., downloaded from a package manager uniform resource locator [URL] at a particular point in time, or pulled from a specific supplier distribution server with supplier signatures)? Legacy systems often lack chain of custody to establish verifiable point of origin for open source components that were brought in by developers or received from commercial suppliers with no signatures on the software. Chain of custody can be re-established by sourcing components from an authoritative point of origin.

   For example, an organization's software development pipeline may require a dependency manifest that lists components in an enterprise component repository (or package manager). The component repository contains basic metadata necessary for an SBOM (supplier, product name, and version). This metadata may also include component hashes and point of origin (e.g., public package manager or open source URL). Acquisition of SBOM component data from a package manager is preferable to acquisition of SBOM component data from a document file (e.g., spreadsheet) that describes the component composition at a point in time, because the component composition may have changed since that document was produced.

   Legacy processes often involve manual or semi-automated curation of SBOM data from existing platforms and processes that may resolve to spreadsheets that are manually maintained or exported from systems whose data is manually maintained. Lists of a software capability's open source components may be maintained for copyright compliance purposes. These OSS or "FLOSS" (free/libre OSS) lists may be leveraged for SBOM generation if no other data is available, but the absence of hashes means that there is no verifiable link between a component name and the actual component used in the software capability. In order to ensure software integrity, some verification process is required, but copyright compliance spreadsheets are a starting point. Binary scanning may be used to ensure software integrity.

2) Software Identification: Are software components identified with authoritative identities (e.g., Common Platform Enumerations [CPEs], package URLs [PURLs], SWID tags) that will map to vulnerability information? Many legacy systems identify components with names that were manually designated when software was brought into the organization. These in-house names can be used in an SBOM, but will ultimately need to be resolved and authoritatively identified, either by the SBOM producer or the SBOM consumer, to enable vulnerability management.

More recently proposed are the identifications or URLs of software in package management, such as public repository URLs. Such an approach works well for software covered by (public) package management services, which is usually not the case for proprietary components. It is important to note that the package URL specification designates globally unique public identifiers—the location of a package in a publicly accessible package manager, as opposed to a corporate internal package manager whose namespace diverges from the public package manager namespace. It is not simply a concatenation of supplier, package, and version information present on an internal system. To adhere to the PURL specification, suppliers using PURL as a software identifier should be using PURLs that associate a component with a public package manager location when the point of origin is a publicly accessible package manager. For proprietary or first-party packages whose point of origin is the supplier, use of an internal package repository qualifier is acceptable.

Time of Generation for an SBOM not Created at Build Time

For SBOMs generated post-build, the SBOM should include time-of-generation (i.e., when the SBOM was authored). Versioning for the SBOM itself may be included in the title or body of an SBOM. Post-build SBOMs may also be signed to ensure integrity of the SBOM and verification of the author of the SBOM.

Deliverable: What's in the Box

It is important to understand what is and is not described by an SBOM. For instance, if an application is delivered as source or as a compiled binary with an SBOM that enumerates software dependencies that are incorporated into the application, the SBOM pertains to that application. This is a relatively simple case.

If that application installs runtime dependencies, operating systems, dynamic link libraries (dlls), updaters, shared libraries, or other inclusions, the SBOM's information should resolve ambiguity about what it describes in a way that is sufficiently explicit for the consumer to understand that additional components or utilities delivered with the application are—or are not—enumerated in the SBOM. For instance, a commercial software product might be delivered with an SBOM for an application and a separate SBOM for its installer or distribution package. This question of coverage applies to container images and container dependencies as well. Similarly, the SBOM of the tool chain used to create the software may be provided.

What is important to avoid is a false-negative gap, where the supplier supplies an SBOM for their application, but the installer deploys other software that is not in the SBOM because those components are not "in" the application, even though they are delivered or installed with the application. Runtime dependencies can be a significant source of vulnerability, and they should not be a blind spot for either the supplier or the consumer if they're not included in a binary. There have been cases where the installer itself-—-not what it installs—is infected with malware (or is malware). So the importance of clarity for “known knowns”—what is the outer limit of what an SBOM describes—is security-relevant.

External Services: Crossing Trust Boundaries

System assurance requires visibility of external services that cross trust boundaries, e.g., calls from an application to an Internet service to provide data required by an application. Automated update services are a security-relevant external service.

If a software capability relies on call-outs to external services to perform, SBOMs may be used to enumerate external services required by a software deliverable. Enumeration of external services may or may not be necessary for acceptance by software consumers, subject to their own security and regulatory requirements. This set of requirements is evolving, and is not currently part of the SBOM baseline.

From a supplier’s perspective, the enumeration of external service calls can streamline customer assurance, acceptance, and authorization of software by simplifying dynamic testing and resolving network-detection alarms regarding call-outs. It can also rationalize the establishment of network permissions necessary for a software capability to function.

Other SBOM Considerations

Not all SBOM decisions relate to the technical creation of the dependency graph and related data. Tracking and sharing software supply chain data also involves operational, business, and even legal questions. Below are several issues that an organization may want to consider.

Intellectual Property/Confidentiality of SBOMs

Suppliers may regard SBOMs as competitive information and may not want their SBOMs to be publicly distributed. Because SBOM information must be able to flow through intermediary consumer-suppliers to their end-consumers (i.e., from a subcontractor to a contractor to a customer) the appropriate confidentiality regime is to treat the SBOM as proprietary information subject to contractually negotiated agreements, rather than to use the copyright status of an SBOM to preclude distribution.

For instance, an SBOM provided with a Creative Commons CC0 license, which is best practice for SPDX generation and provision, can be provided under a nondisclosure agreement. It does not require or allow sharing outside the bounds of negotiated nondisclosure agreements, but it does enable the transfer of data within the permission boundaries of negotiated agreements.

For additional references to work on the provision and exchange of SBOMs by NTIA’s Framing Group, please review Sharing and Exchanging SBOMs.

Validation of SBOM Format

There are several tools to check that an SBOM format is valid (e.g., necessary fields are present, and the document is properly structured). SBOM validation is specifically a check on the correctness and completeness of the format, rather than the quality or accuracy of the data used to populate SBOM fields. SBOM validation may be performed before delivering an SBOM, to ensure that SBOM consumers will be able to parse an SBOM in a predictable, error-free, and automated fashion.

Examples of SBOM validation tools include:

- SPDX Online Tool validates SPDX format SBOMs: https://tools.spdx.org/app/validate/.
- SWID Tools: https://pages.nist.gov/swid-tools/swidval/.
- CycloneDX CLI Tool and Web Tool validates CycloneDX format SBOMs: https://github.com/CycloneDX/cyclonedx-cli/; https://cyclonedx.github.io/cyclonedx-web-tool/.

Verification of Components

For those cases where the SBOM consumer (or the SBOM creator) is concerned with verifying the information in an SBOM, there exist frameworks for assessing the maturity of the creation. These include Open Web Application Security Project (OWASP) Software Component Verification Standard (SCVS) and ISO 5230.

OWASP SCVS (https://owasp-scvs.gitbook.io/scvs/) is a framework for baseline assessment of SBOM production capacity and SBOM completeness beyond the baseline elements. This standard allows organizations that produce SBOMs to gauge their technical maturity, and the level of verification (and therefore trust) that they can place in the underlying data that is used to populate an SBOM.

OpenChain (https://www.openchainproject.org), also known as ISO/IEC 5230:2020, is a process management standard, designed to explicitly address the key inflection points necessary for accurate identification and tracking of software inbound, internally, and outbound. As a process management framework, ISO/IEC 5230 is SBOM-standard agnostic. This allows organizations to leverage pre-existing activities, and to adopt the specific SBOM format (for example SPDX) and encoding (JSON, XML, etc.) best suited to their requirements. The formulation of ISO/IEC 5230 ensures that whatever SBOM is being used in production can be adequately verified at each critical phase of production. For the same reason, it will expose the maturity and path to improvement for current organizational process management.

Binary analysis may also be used to identify or verify binary components in an SBOM. Software composition analysis (SCA) tools may be used to achieve transparency on software composition.

References

The Minimum Elements for a Software Bill of Materials (SBOM)
https://www.ntia.gov/files/ntia/publications/sbom_minimum_elements_report.pdf
Framing Software Component Transparency: Establishing a Common Software Bill of Material (SBOM)
https://www.ntia.gov/files/ntia/publications/framingsbom_20191112.pdf
GITHub lfscanning
https://github.com/lfscanning
CycloneDX examples
    GitHub - CycloneDX/sbom-examples: A repository with examples of CycloneDX SBOMs from various projects
GITHub Package URL descriptions
    GitHub - package-url/purl-spec: A minimal specification for purl aka. a package "mostly universal" URL, join the discussion at https://gitter.im/package-url/Lobby
Sharing and exchanging SBOMs
    https://www.ntia.gov/files/ntia/publications/ntia_sbom_sharing_exchanging_sboms-10feb2021.pdf
SPDX validation tool
https://tools.spdx.org/app/validate/
NIST Software Identification tool
https://pages.nist.gov/swid-tools/swidval/
CycloneDX web validation tool
https://github.com/CycloneDX/cyclonedx-cli/
CycloneDC web validation tool
https://cyclonedx.github.io/cyclonedx-web-tool/
ISO 5230 OpenChain Specification
ISO/IEC 5230:2020(en), Information technology — OpenChain Specification
OWASP SCVS
https://owasp.org/www-project-software-component-verification-standard/
Openchainproject.org
https://www.openchainproject.org/
OpenChain
https://www.iso.org/standard/81039.html