Playbook: Software Suppliers

SBOM Production

|  |
| --- |
| Draft version 1.0 2020-01-22  This document is available at:  [https://docs.google.com/document/d/16FwpPX3P0Pd1D82Dp2VmpRnaMWUA-wvfXbL7AIXDthM/](https://docs.google.com/document/d/16FwpPX3P0Pd1D82Dp2VmpRnaMWUA-wvfXbL7AIXDthM/edit#) |

# 

[Overview](#_buhjqaz63gsd)

[Summary of Workflow](#_px6ds6dyh3o)

[Relevant Workflow Differentiators](#_7gf0bjv4fg7y)

[Current Best-Practice vs. Legacy Engineering Processes](#_3qf5744j2trb)

[Containers - export of SBOMS from a containerization process](#_z91depyh3763)

[Time of Generation - Code vs Build vs Assemble](#_qx5zh24a46af)

[Deliverable: What’s in the Box](#_vu2f71228wfz)

[External Services: Crossing Trust Boundaries](#_v3xpsp5ntyy3)

[Other SBOM considerations](#_tok19xwqpdyz)

[IP/Confidentiality of SBOMS](#_u0bwmvnwm5wu)

[Verification of SBOMs](#_2m5ia1ffgu15)

[Hardware and non-monolithic systems](#_bu03nwp7lao2)

# 

# Overview

This playbook outlines workflows for the production of Software Bills of Materials (SBOM) production and 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 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. Much of this document helps characterize the considerations and decisions that teams will make, based on their existing tools and processes and their level of technical maturity. However, we can roughly characterize the SBOM production process in the following steps:

1. Identify software components included in a software capability.
2. Acquire data about components used in a software capability.
3. Import component data into structured SBOM formats.
4. Validate SBOM to ensure that format is valid and minimal required information is 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 towards the efficient and effective creation of SBOMs.

### Current Best-Practice vs. Non-Automated Engineering Processes

**Automated Workflows: Production of SBOMs as a Build Artifact**

Current best-practice engineering workflows, including Git for software version control and Continuous Integration/ Continuous Delivery (CI/CD) pipelines enable automated collation and generation of SBOMS from a pipeline build. Because these processes are an automated process that derives 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 then 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 “common core” element names, identified in the [Framing document](https://www.ntia.gov/files/ntia/publications/framingsbom_20191112.pdf) to the SBOM specific data elements that have been identified to represent these common elements. SBOM suppliers are expected to follow these common element mappings when creating their SBOM artifacts.

* + Functional workflow (tool-agnostic) for commit → build with SBOM production as an output
  + SPDX - <https://github.com/lfscanning> has SBOM’s for some common open source projects.
  + CycloneDX: Some examples are here: <https://github.com/CycloneDX/cyclonedx-core-java>
  + SWID?

**SBOM production in build pipelines and software factories**

Many organizations automate the creation of SBOMs in their development lifecycle. 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 plugins. These plugins 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. The availability of Jenkins libraries and GitHub actions that perform automatic SBOM creation are such examples.

### 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/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 including the operating system, OS packages, applications and their associated libraries along with other artifacts that may have been incorporated into various layers. SBOMs that describe container images must include the aggregation and accurate identification of all software from all layers.

Multiple container images may be used together to form an application. The orchestration of 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](#_3vggxo663tax)

SBOMS generated at build-time should include the time of build as part of the SBOM as indicator that the SBOM was generated at build-time.

Signature: For integrity, SBOMs generated as part of a build process should be signed as part of the build process.

**Post-Build SBOMs**

Not all the delivered software can be part of build-time SBOM generation. Many software capabilities, especially software 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 an 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.

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 got to where it is. Salient questions include:

1. Chain of Custody: 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. download from a package manager 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 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 instance, an organization’s software compilers 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” 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.

1. Software Identification: Are software components identified with authoritative identities (e.g. CPEs, PURL, 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](https://github.com/package-url/purl-spec) designates globally unique public identifiers - the location of a package in a publicly accessible package manager, vs. 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, for components whose point of origin is a publicly accessible package manager. For proprietary or first party packages whose point of origin is the supplier, internal package repository identifiers are 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.

Signatures

Post-build SBOMs may be signed to ensure integrity of the SBOM and attestation of the author of the SBOM.

### Deliverable: What’s in the Box

It is important for customers 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, dlls, installers or other inclusions with an installer, the SBOM’s information should resolve ambiguity about what that SBOM describes in a way that’s sufficiently explicit for the consumer to understand that additional components or utilities delivered with the application are - or are not - enumerated in the SBOM. This question of coverage applies to container images and container dependencies as well.

What’s 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 aren’t “in” of the application, even though they are delivered and 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 envelope that SBOM describes - is security-relevant.

### External Services: Crossing Trust Boundaries

Customer 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 external services, 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 is a set of requirements that is evolving.

# Other SBOM considerations

Not all SBOM decisions are around 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 a summary of issues that an organization may want to consider.

### IP/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 confidentiality regime appropriate to the protection of SBOM confidentiality is to treat the SBOM as proprietary information subject to contractually negotiated nondisclosure 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 under NDA. 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.

Reference to work on the provision and exchange of SBOMs by NTIA’s Framing Group:

<https://www.ntia.doc.gov/files/ntia/publications/ntia_sbom_framing_sharing_july9.pdf>

### Validation of SBOMs

Make sure that the data format is valid, consumable, etc

Verification of Components

For those cases where the SBOM consumer (or the SBOM creator) is concerned with verifying the information in a SBOM, there exist frameworks for assessing the maturity of the creation and the processes such as OWASP SCVS (owasp.org/scvs) or [ISO 5230](https://www.iso.org/obp/ui#iso:std:iso-iec:5230:ed-1:v1:en) ( guidance on how to apply can be found at the [OpenChain website](https://www.openchainproject.org/))

Reference to OWASP SCVS as a framework for baseline assessment of SBOM production capacity

Comprehensive nature of the data

Completeness of the DAG

Level of detail of for each of the components (Min Viable vs more info)

<https://owasp-scvs.gitbook.io/scvs/>

<https://owasp.org/www-project-software-component-verification-standard/>

ISO/IEC 5230 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, allowing organizations to leverage pre-existing activities, or to adopt the specific SBOM (for example SPDX) most 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.

Punch List

1. Automated workflows - bottom of page 3 - need verbiage about export of SBOMs from a CI/CD pipeline that’s format-agnostic guidance. To make sure that it’s format-agnostic, can Steve and Kate work up a paragraph that describes SBOM export with a binary build in a way that’s format agnostic?

2. Containers - export of SBOMs from a containerization process (ditto).

3. Under Software Identification, I’ve explicitly called out the PURL specification as referring to the location of packages in publicly accessible package managers for packages whose point of origin is a publicly accessible package manager. For proprietary or first party packages whose point of origin is the supplier, internal package repository identifiers are acceptable.

4. External services - have assigned to Steve and Kate a description of how CycloneDx and SPDX handle external services.

5. Time of Generation - this is (IMO) the last piece that we really have to chew and clarify. Happy to have you guys take a pass at it. We have to accommodate embedded use cases and at speak to the system-of-systems reality of software capabilities as delivered. I suggest we tuck any discussion of hardware or non-monolithic systems into this section, while constraining the scope of that discussion.

6. Validation of SBOMS - need to point out that tooling for this exists (

7. Verification of components: if Steve could add some color and detail to the SCVS section that’d be great - this is a good way for the supplier to gauge their “known unknowns” vs. “unknown unknowns” - and frankly, whether their “known knowns” are actually known.

Backlog:

* Use case questions
  + IP management use cases may care about snippets
  + Stack Overflow and other copy-and-paste
  + Operating System SBoMs and how they can be referenced by container users