DRAFT PLAYBOOK for Software Consumers: SBOM Acquisition, Management and Use

This document is available at:
https://docs.google.com/document/d/1Ae0l1MDS8m1on58e8mdVIA9NujzPD0k5j352VlDZr9l/edit

This playbook outlines workflows for the acquisition, management and use of Software Bills of Materials (SBOM) by software consumers. Software consumer is broadly defined to include commercial and non-commercial entities acquiring third party software capabilities from a supplier. A supplier is defined to include:

- Commercial software vendor
- A contract software developer supplying a software deliverable
- An open source software supplier maintaining source code in an open source repository, binary artifacts in a package manager, or both.

SBOM acquisition from managed service providers (MSPs) or SaaS services is outside the scope of this document, since the delineation of inclusion criteria for an SBOM from MSPs is not well-defined, and because the information required to adequately manage supply chain risk from MSPs includes system-level configurations outside the bounds of an SBOM that describes the codebase.

Commented [1]: Cross check w Framing doc. See the ongoing need for a glossary

Commented [2]: what is the difference between these? Isn't a contract developer just part of a software vendor?

Commented [3]: in crawl walk run, the distinction is flying (and personally I still don't see it) and adds confusion I think would be best left out at this level

Acquisition of SBOM from Supplier

SBOM acquisition includes:

- Contractual procurement of a commercial product
- Download of commercial closed-source software
- Contractual procurement of professionals services that include the development and delivery of software capabilities
- Acquisition of open source software applications or components for internal deployment
- Acquisition of open source software applications or components for internal development is discussed in the Supplier/Consumer playbook

SBOM Coverage for Software Systems

Software deliverables covered by SBOMs vary in scope and complexity. These include:

- A single application with internal dependencies but no install-time dependencies
- An application with install-time dependencies and internal dependencies

- Software containers
- Software systems with multiple endpoints

The specification of requirements by consumers should include coverage for all software components delivered by a supplier, whether these are internally incorporated components of a software application, run-time dependencies or the contents of a container.

Vendor provision: An SBOM may be provided by a vendor either before delivery of a software product, as part of the customer pre-procurement process, with the product at time of delivery, and/or with updates to the product as updates are delivered to the customer. This document specifically excludes SaaS in the definition of product delivery, and excludes dependencies on vendor or third-party SaaS as an attribute of a product SBOM. Extensions of this document and the SBOM scope will likely include SaaS dependencies of installed software as a domain of data enumerated by an SBOM. But in the context of this document, a vendor-provided SBOM is defined to include the contents of a software deliverable that is transferred from a supplier to a consumer.

In order to represent a complete bill of materials that creates sufficient transparency for software asset management and vulnerability management by the consumer, a vendor SBOM must enumerate third party software dependencies (both open source and proprietary) incorporated into the vendor product, as well as installed dependencies required at run-time. For instance, if a vendor download manager installs software drivers, libraries or open source applications along with the vendor's binary application, a complete SBOM for that product would include third party dependencies within the compiled binary as well as software installed by the download manager. In short, if a vendor product installs a third party dependency on the consumer's system, either as a constituent component of the vendor product or as an installed enabling capability, it must be enumerated in the SBOM in order to provide requisite transparency for software asset management and vulnerability management. The alternative of failing to list installed run-time dependencies leaves a dangerous gap in situational awareness that exposes the consumer to compromise when an outdated and vulnerable installed dependency is exposed and a remediated update has not been supplied by the vendor.

Commented [4]: if vendor A causes to install an OS dependancy B, is that included? For instance, some CRM system, when installed on Redhat, installs the Redhat Apache2 package. I think it should list apache2, because the operator didn't install it themselves, and wouldn't think that they were running it otherwise. Redhat's apache SBOM might be where the openssl dependancy then gets listed.

The delivery of vendor products in software containers is increasingly common and presents a similar scenario. The vendor product may not include vulnerable dependencies, but enabling software installed in the container may be vulnerable and exploitable. For this reason, a manifest of container contents should be provided by vendors as part of an SBOM data workflow. A container SBOM, listing the container's contents including vendor application(s), may be provided as a separate item than the SBOM for the vendor product that includes the product's constituent components.

Contractor provision: For software deliverables provided by a supplier who is developing or integrating these capabilities under contract from the consumer, the intellectual property status of contractor-developed software and software systems as bespoke deliverables, and the terms and conditions of contracts for professional services, give consumers more latitude to

require and enforce software transparency and SBOM provision than exist within the contractual framework of vendor product acquisition. Specifically, the generation and update of SBOMS is a service than can be specified and required under the terms of a master services agreement for software development, Statements of Work under that agreement and flow-down provisions to subcontractors within the scope of an MSA.

For source code delivered by a contractor, an SBOM should accurately reflect all dependencies required to compile the code, to ensure the consumer can compile the code and take full delivery of the contracted capability. For source code deliverables, the consumer may also choose to compile the code on the consumer's infrastructure to verify the accuracy and completeness of the SBOM, and may, as part of that build process, generate a built-time SBOM as an automated process for cross-comparison with the supplier SBOM or to provide more complete data that includes build-time metadata.

For binaries delivered by a contractor, SBOM acquisition is similar to vendor SBOM acquisition. However, the consumer may require more complete data in a contractor binary SBOM, such as build-time metadata and authoritative supplier and product identities for proprietary third party components that are unlikely to have discoverable public identifiers. Under the terms and conditions of a professional services contract, the consumer may also reserve the right to reverse-engineer a binary deliverable to verify its composition against the SBOM provided by the supplier, and to investigate and resolve any discrepancies in a mutually agreed-upon manner.

Open Source Software: While some open source software projects generate SBOMs as part of their distribution process (e.g. as an information asset available for binary packages and distributions), this practice is not, and is not likely to be, universal. Likewise, open source code repositories may include formatted SBOMs, but the purpose of a source code repository - to enable development by one or more contributors - inherently requires enumeration of dependencies in manifests such as requirements.txt or pom files that are necessary to compile the software. In the absence of a formatted SBOM (e.g. SPDX, CycloneDX or SWID), the consumer may derive an SBOM from an open source code repository, either by implementing automated generation of an SBOM in the consumer build pipeline or by using an open source or commercial capability to generate an SBOM from the repository.

It is important to note that the composition of binary packages may differ in security-relevant ways from the open source code that package points to as its contents, because additional dependencies are often included when the binary package is compiled. For this reason, it is important to understand the point of origin for open source components, and to realize that ambiguity about point of origin - source vs. binary - should be resolved by assuming that the component has the more common higher-risk profile of a binary package, which may have additional dependencies and vulnerabilities.

SBOM Ingestion and Parsing

There are two scenarios for what happens after an SBOM is acquired.

Archival storage of SBOM files: The first scenario is that a consumer lacks the capacity, the intention or any regulatory requirement to make use of the information. If the SBOM is supplied but does not feed any kind of enterprise workflow, the playbook ends with content management: when the SBOM is received, where is the file stored and what are the content management, data retention and life cycle policies for the storage of that file. This may entail storage in an enterprise inventory or IT asset management database (e.g. the same database or file system that stores the serial numbers of computers or furniture).

If specific security measures for the storage of supplier SBOM information are contractually specified, the consumer must ensure that the security controls for systems into which SBOM data flows meet or exceed controls specified the terms and conditions of SBOM provision by a supplier.

Life cycle policies for storage of SBOMs as files should correlate to the life cycle of the software deliverable represented by that SBOM. The file should persist until and unless the consumer has provably decommissioned that software asset, i.e. automated scans or manual inventories indicate that the software asset corresponding to the SBOM are no longer present or installed on the consumer's infrastructure, and the information is no longer relevant for legal and forensic purposes (e.g. discovery of a breach that occurred before a software asset was updated or removed).

Decommissioning - verification that an asset has been removed from a system or facility - is an order of magnitude more difficult than deploying assets. It requires a higher level of transparency and positive control than most IT enterprises possess. For this reason, especially given the relatively small size of SBOMs, the default life cycle policy for SBOMs should be archival retention.

The second scenario is that data from SBOMs will feed enterprise workflows, including procurement, asset management, vulnerability management and over-arching supply chain risk management and compliance functions. In this scenario, the SBOM is less useful as a file than as a collection of data that can be parsed, extracted and loaded into automated processes or systems of record. There are three options for SBOM ingestion and parsing for enterprise ETL:

1) Internally developed tooling (i.e. scripts). In enterprises that have the technical resources or the desire to internally develop ETL processes for SBOMs, methods such as Python scripts may be used to extract and load data from specified SBOM fields into external data platforms and workflows.

2) Open Source Software tooling for extraction of data from SBOMs

Commented [5]: @kstewart@linuxfoundation.org ? @steve.springett@owasp.org ?

Commented [6]: Ok to refer to the taxonomy? I'm guessing you're looking a couple of sentences, rather than listing of tools, etc.

Commented [7]: Yup!

3) Commercial tooling: Increasingly, commercial software packages and platforms ingest and resolve SBOMs. These include GRC and vendor management platforms, configuration management systems, software supply chain assurance platforms and software asset management systems. Sector specific commercial solutions, such as those used in medical devices and fleet management, are on the adoption curve for SBOM ingestion. Enterprise customers who already have these systems installed should coordinate with platform vendors to validate whether their installed solutions ingest SBOMs and if not, what is the roadmap for doing so. Because capabilities in this field are emerging rapidly, cataloguing commercial solutions that support SBOM ingestion is not in this playbook but may be separately maintained by the NTIA Formats and Tooling working group with verification of vendor attestations re: SBOM ingestion and resolution.

Extraction from formats such as SPDX, CycloneDX and SWID is relatively straightforward based on field names. However, these methods do not enable full resolution of SBOM contents if components are not authoritatively identified, are erroneously identified, or if the data provided is incomplete. Since many suppliers do not maintain chain of custody or produce SBOMs using automated and verifiable methods from a software build, some level of entity resolution will usually be required (see below). Resolution of transitive dependencies for an SBOM that only enumerates direct dependencies requires software entity resolution.

Software Entity Resolution

There are a number of scenarios in which software entity resolution is required to authoritatively identify components of an SBOM, so that components can be accurately mapped to vulnerabilities and disambiguated in configuration management and software asset management workflows. In order from easy-to-difficult, these scenarios include:

SBOM auto-generated as the artifact of a version-controlled software build: SBOMs that are automatically generated as an artifact of a version-controlled software build are relatively clean, in the sense that the names of dependencies are assigned by a machine workflow that identifies components that are specified to meet the requirements of software compilation. This scenario minimizes the arbitrariness of human-in-the-loop software naming, since the names of software projects are likely to be replicated with fidelity from point of origin to the build process, whether directly or in an indirect process that entails transfer from point of origin to an enterprise component repository, and from the enterprise component repository to the build process. Version specification as an element of software governance, e.g. the requirement to specify a particular version of dependencies required for a software build, enables specificity and disambiguation of SBOM contents.

SBOM produced outside a software build: SBOMs produced outside a software build, especially those collated from legacy software development processes and platforms of record, generally require additional software entity resolution of component identity. Manual and arbitrary naming conventions may be ambiguous or inaccurate. Even where software components accurately

reflect the names commonly used by developers, these names may not correspond to the software identifiers used in vulnerability data sources such as the National Vulnerability Database or vendor security advisories. NIST's proposed migration from CPEs as software product identifiers to SWID tags does not solve this problem when components are not authoritatively identified at point of origin or in an enterprise system of record. In many cases, there is ambiguity or absence of authoritative information from any source, including the supplier, which entails the creation of pseudo-identifiers that hew to the naming conventions of software identifiers in vulnerability databases.

This entity resolution may be performed manually via lookups, semi-automatically using a combination of scripts and manual adjudication, or automatically by a first-party or commercially licensed software entity resolution engine. Name assignment using aliases or pseudo-authoritative identifiers can be manual or automated and should leverage entity resolution at the vendor and component level to formulate a component name in the absence of authoritative external identifiers.

Where possible, authoritative identifiers should be required for vendor-licensed components, to ensure that these components, which have a contractually designated supplier, are consistently named across software products that include those licensed components. Where possible, commercial SWID tags should be required to attain precise versioning of commercial software, which differs from open source components in its naming conventions for patch and release versions, service packs and updates. The same version of a commercial software capability may differ significantly in its composition and security profile, depending on installed updates.

Chain of Custody and Software Types

Assuming the software components of an SBOM have been accurately resolved from a namespace perspective, there is still ambiguity with regard to the software type and point of origin for accurately named components. Specifically, the contents of a component acquired as open source (uncompiled) code from a source repository such as GitHub may differ in security-relevant ways from a binary package of the same component published to a package manager. Package managers often include runtime dependencies of the source component for which the package is named. Vulnerability databases only map known vulnerabilities to the component itself, as it is represented in the source code repository, vs. the binary package from a package manager that includes vulnerable runtime dependencies in the binary. These binary package inclusions will yield false negatives on a CVE lookup based on the designated SBOM components. Essentially, the packaging materials are not listed on the box, and the packaging materials are vulnerable.

For this reason, especially for high-assurance systems and environments, software transparency requires either that the point of origin for a software component be identified, or that the customer's default assumption for vulnerability management is that designated open source components are have been included in the supplier deliverable as binary packages and therefore may contain third-party libraries and run-time dependencies as packaging inclusions

that are not present in the open source repository to which the binary package manager refers as the compiled component.

This coverage issue is magnified in the delivery of software containers, which often contain operating systems, runtime dependencies and miscellaneous inclusions that are not listed in an application SBOM. SBOMs for containerized software should include all container contents as a top-level manifest, in addition to one or more SBOMS for the software payload whose delivery is enabled by the container.

Risk acceptance of ambiguity regarding source vs. binary packages or containers should be a “known unknown” in software asset and vulnerability management, and should drive chain-of-custody requirements for high assurance software capabilities. OWASP’s Software Component Verification Standard (SCVS) includes a number of controls for disambiguation of components by type and point of origin.

Deleted: SCVS

Transitive Dependency Resolution

If an SBOM is supplied with direct software dependencies only, full software entity resolution entails resolution of those direct dependencies to catalog all transitive dependencies in the software deliverable, because it is often transitive dependencies that become vulnerable, and because the mapping of transitive to direct dependencies in vulnerability databases is not current or complete.

Transitive dependency resolution may be achieved using open source tools or commercial solutions. Unlike the entity resolution of direct dependencies, accurate and complete resolution of transitive dependencies is not tractable using manual processes - the pyramids of nested dependencies are too large and change too fast as underlying components are updated and substituted in the supply chain.

Third Party Processes and Platforms

Once SBOMs are satisfactorily resolved, the data within them is most usefully employed in processes and platforms that include:

- Configuration Management Databases (CMDBs)
- Software Asset Management (SAM) systems
- Security Operations Centers (SOCs)
- Procurement workflows, which may include pre-procurement diligence, contractor/vendor management systems and third party risk and compliance management and reporting.

The transformation of SBOM data from files into data feedstocks for other systems requires enabling processes, including:

1. Auditable custody and storage of SBOM files, e.g. in a registry that records the date of receipt for each SBOM and the SBOM version. For legal and compliance reasons, version and access control to these files should be auditable, and any contractual requirements for confidentiality of proprietary SBOMs should be enforced at the registry level.

2. Extraction, Transformation and Loading of SBOMs into enterprise processes and platforms requires a mapping process to correlate specific components to one or more applications, systems or endpoints. Much of the value of these processes and platforms derives from the mapping and update functionality that maintains the accuracy of software inventories and configurations across an enterprise. In certain ways, SBOM data is similar to attribute data for any software asset. However, SBOM data does differ significantly in its volume and granularity - there is more data and more detailed data per software asset than for a conventional commercially procured software capability with no SBOM. Therefore, the workflow volume, particularly in automated workflows, may scale significantly and enterprises must plan for this.

Ongoing Monitoring

One of the biggest differences and greatest sources of value for SBOMs is the ability for end-users to monitor vulnerabilities in parallel with whatever vulnerability management is conducted by the supplier. This “trust but verify” capability to continuously monitor the vulnerability status of a supplier’s software dependencies creates continuity of assurance by eliminating gaps in situational awareness, including:

- The time it takes a supplier to detect vulnerabilities. This is especially relevant for software capabilities that are not being actively built or maintained. If vulnerability detection is done as part of static analysis in a software development pipeline, and the software isn’t built every day, there can be days, weeks or even months when vulnerabilities are not detected by a supplier that is not actively developing a capability.

- The time it takes for a supplier to remediate vulnerabilities once they’re detected. If a supplier knows there are critical vulnerabilities and is working on updates, but does not inform the consumer that these vulnerabilities exist, there is no opportunity for the consumer to implement compensating controls while the supplier works on remediation.

- The time it takes for a supplier to ship a remediated update. Even when security issues are resolved, the release schedule of a supplier may create delays in the delivery of a remediated release, to maintain a scheduled cadence or enable the delivery of new features “with security updates.” With no SBOM, the consumer is in the blind and unable to implement compensating controls or make risk acceptance decisions before a remediated update is released.

Parallel or out-of-band monitoring of SBOM components creates situational awareness on the consumer side, and raises the level of transparency into security remediations that suppliers might make to vulnerable components, which can be enumerated in a Vulnerabilities and Exploitability (VEX) file that accompanies an SBOM.

Higher levels of transparency can enhance security. But increase in security posture entails some business process and cultural change on the consumer side. Specifically, with regard to steps a responsible supplier may have taken to remediate security issues in vulnerable dependencies without removing those dependencies, e.g. by limiting function calls or removing subcomponents, it is important for consumers to acknowledge and accept valid remediations enumerated in a VEX file instead of demanding a "clean scan" that does not contain any CVEs. This is a shift from compliance to risk management, and unless that shift occurs, enterprise security stakeholders will be buried in findings that are not legitimate and are impossible to resolve.

IP and Confidentiality Status of SBOMs

The intellectual property status and confidentiality of an SBOM may vary depending on whether it is generated and/or provided by a vendor, a contractor, an employee or an open source development community. But in all cases the intellectual property and confidentiality status of SBOM should be explicit and preferably enumerated within the SBOM itself.

Confidentiality

For vendor and contractor software deliverables, the confidentiality terms that apply to the SBOM, as distinct from the software deliverable itself, should be explicitly defined in the terms and conditions of a software license or master services agreement. From the software consumer perspective, the consumer should, without limitation, be allowed to use an SBOM for internal purposes and to be able to share the SBOM with any third party entity which is subject to the same confidentiality and/or non-disclosure terms as the consumer, with regard to the SBOM. For instance, a vendor or contractor retained by the consumer under NDA to assess and assure software assets should not be precluded from access to SBOM data because they are not direct employees of the consumer.

Likewise, seat-licensing restrictions that apply to use of a software product should not apply to SBOMs as a form of digital rights management. SBOMs generated and supplied by software development contractors performing professional services should be intellectual property transferred to the consumer under the same terms as conditions as the software deliverable(s), whether that status is a non-exclusive perpetual royalty-free license or work for hire.

SBOMs provided by open source software suppliers, whether as part of an open source software project with no financial relationship to the consumer or as part of commercially

Commented [8]: @kstewart@linuxfoundation.org - do a detailed review pass on this section.
Assigned to Kate Stewart

Commented [9]: @afriedman@ntia.gov - please add in recommendation about contracts putting restriction, rather than using data license.
Assigned to afriedman

Commented [10]: In general, we recommend access control and usage restrictions should be specified in contracts and policy, rather than intellectual property and licenses.

Commented [11]: should we consider recommending how to mark confidentiality if it is going to be marked? Would TLP be appropriate https://www.cisa.gov/tlp

supported open source software, should be explicitly licensed under an open source license. This license does not necessarily need to correspond to the open source license of the software itself. For instance, a software project delivered with an MIT license may have an SBOM delivered with a Creative Commons license. The default license for SPDX is CC 0.

Commented [12]: A default license would be useful sense in order to avoid a heterogenous license landscape for the SBOM of a system resulting from differently licensed SBOMs of sub systems

SBOM IP/Confidentiality Status and Intermediary Suppliers

If the consumer is an intermediary supplier - an entity that receives SBOMs for software that is incorporated into another software deliverable with its own SBOM, care should be taken to confirm that SBOM information incorporated into the intermediary supplier's SBOM provided to downstream consumers meets the licensing terms and confidentiality conditions of the SBOMs supplied to the intermediary supplier, to prevent the intermediary supplier's SBOM from violating either the license or the confidentiality provisions of the SBOMs from which the intermediary supplier's SBOM was derived.

For instance, if a medical IT system includes devices with confidential SBOMs, the license and confidentiality provisions of the IT system supplier's agreements with its device suppliers should convey the right to supply the SBOM information in those devices to a downstream customer (e.g. a hospital system), subject to congruent confidentiality requirements . If those rights do not convey to downstream consumers, the IT system supplier must be able to authoritatively identify the device components of its system to the consumer, who may then request device SBOMs from device suppliers under separate confidentiality agreements.

It is simpler and more expedient for intermediary suppliers to acquire SBOMs whose intellectual property status conveys the right to share the SBOM with downstream consumers subject to congruent confidentiality terms and conditions, and this is the ideal default for SBOM acquisition by intermediary suppliers.

Under no circumstances should SBOM IP and/or confidentiality status preclude the identification of a software component by an intermediary supplier. There should be no "gag rule SBOM" that allows a supplier to require the exclusion of its software from an intermediary supplier's SBOM and/or fail to signal that components subject to these conditions are included in an intermediary supplier's software capability. If such rights were to be asserted and successfully enforced, it would hobble the transparency required for effective software supply chain risk management and trigger the proliferation of "unknown unknowns" that compromise security posture.