Types of Software Bill of Material (SBOM) Documents

Introduction

Today there is a widely used definition of the minimum content of a Software Bill of Material (SBOM).[1] However, an SBOM may contain different forms of the minimum information sourced from different product artifacts. Given the disparate ways SBOM data can be collected, tool outputs may vary and provide value in different use cases. This document summarizes some common types of SBOMs that tools may create today, along with the data typically presented for each type of SBOM. An SBOM document may combine information for multiple SBOM types.

Definitions and Discussions

The following two tables summarize the different types of SBOMs and the benefits and limitations of each type. This list of SBOM types is not intended to be tightly tied to the software lifecycle. Some SBOM types may be available and useful across multiple lifecycle phases, while others may be available only in one lifecycle phase. Also, the data presented within an SBOM type may vary, depending on the software's lifecycle phase and industry.

Table 1: SBOM Type Definition and Composition

SBOM Type	Definition	Data Description
Design	SBOM of intended, planned software project or product with included components (some of which may not yet exist) for a new software artifact.	Typically derived from a design specification, RFP, or initial concept.
Source	SBOM created directly from the development environment, source files, and included dependencies used to build an product artifact.	Typically generated from software composition analysis (SCA) tooling, with manual clarifications.
Build	SBOM generated as part of the process of building the software to create a releasable artifact (e.g., executable or package) from data such as source files, dependencies, built components, build process ephemeral data, and other SBOMs.	Typically generated as part of a build process. May consist of integrated intermediate Build and Source SBOMs for a final release artifact SBOM.
Analyzed	SBOM generated through analysis of artifacts (e.g., executables, packages, containers, and virtual machine images) after its build. Such analysis generally requires a variety of heuristics. In some contexts, this may also be referred to as a "3rd party" SBOM.	Typically generated through analysis of artifacts by 3rd party tooling.
Deployed	SBOM provides an inventory of software that is present on a system. This may be an assembly of other SBOMs that combines analysis of configuration options, and examination of execution behavior in a (potentially simulated) deployment environment.	Typically generated by recording the SBOMs and configuration information of artifacts that have been installed on systems.

[1] https://www.ntia.doc.gov/files/ntia/publications/sbom_minimum_elements_report.pdf

Runtime	SBOM generated through instrumenting the system running the software, to capture only components present in the system, as well as external call-outs or dynamically loaded components. In some contexts, this may also be referred to as an “Instrumented” or “Dynamic” SBOM.	Typically generated from tooling interacting with a system to record the artifacts present in a running environment and/or that have been executed.

Table 2: Understanding the Benefits and Limitations of SBOM Types

SBOM Type	**Benefits**	**Limitations**
Design	- Highlight incompatible components ahead of licensing purchase or acquisition. - Defines approved or recommended included component list for developer use.	- This may be very difficult to generate. - Unlikely to identify as much detail as found in other SBOM types.
Source	- Provides visibility without access to build process. - Can facilitate remediation of vulnerabilities at the source. - Can provide a view into the dependency tree / hierarchy of the included components.	- Can highlight components (which might have vulnerabilities) that never run or are compiled out in deployed code. - Depending on language/ecosystem, may not include runtime, plugin, or dynamic components, like appserver or platform libraries. - May require references to other SBOMs for completeness..
Build	- Increases confidence that the SBOM representation of the product artifact is correct due to information available during the build and/or Continuous Integration/Continuous Deployment (CI/CD) processes. - Provides visibility into more components than just source code. - Increased trust by enabling signing of the SBOM and product artifact by the same build workflow.	- Potentially have to change the build process to generate this SBOM. - Highly dependent on the build environment in which the build is executed. - May be difficult to capture indirect and/or runtime dependencies. - May not contain the correct versions of dynamically linked dependencies (as they may be replaced at runtime depending on language/ecosystem).
Analyzed	- Provides visibility without an active development environment, such as legacy firmware artifacts. - Does not need access to the build process. - Can help verify SBOM data from other sources. - May find hidden dependencies missed by other SBOM type creation tools.	- May be prone to omissions, errors, or approximations if the tool is unable to decompose or recognize the software components precisely. - May depend on heuristics or context-specific risk factors.
Deployed	- Highlights software components installed on a system, including other configurations and system components used to run an application.	- May require changing install and deploy processes to generate. - May not accurately reflect the software’s runtime environment, as components may reside in inaccessible code.
Runtime	- Provides visibility to understand what is in use when the system is running, including dynamically loaded components and external connections. - Can include detailed information about whether components are active and what parts are used.	- Requires the system to be analyzed while running, which may require additional overhead. - Some detailed information may be available only after the system has run for a period of time until the complete functionality has been exercised.

Conclusion

These definitions are meant as a starting point for clarifying SBOM types that varying tooling types and methods may create. Different tooling approaches may be required to create the same SBOM type for different kinds of software. This document may evolve as the innovation around SBOMs and their uses may require the addition of more SBOM types. Progress in adopting and refining Vulnerability Exploitability eXchange[2] (i.e., VEX), service dependencies, and “SBOM of SBOMs,” among others, may require additional types of SBOMs.

If you would like to learn more about tooling associated with SBOMs, reach out to SBOM@cisa.dhs.gov.

[2] See https://ntia.gov/files/ntia/publications/vex_one-page_summary.pdf for an initial overview. More information will be available at https://www.cisa.gov/sbom.

About this document:
This document was drafted by a community-led working group on SBOM Tooling and Implementation, facilitated by CISA. It is not an official US government document.

Document drafting and preparation for CISA publishing was led by Kate Stewart (Linux Foundation) and Melissa Rhodes (Medtronic).

Pete Allor, Red Hat
Tom Alrich, Tom Alrich LLC
Scott Armstrong, Interos
Sridhar Balasusubramanian, NetApp
Andrew B. Bartels, Operant Networks
Jeffrey Brown, GEHC
Jean Camp, Indiana University
Anesu Chaora, Indiana University
Emily Fesnak, Deloitte
Brian Fox, Sonatype Inc.
Joyabrata Ghosh, Elektrobit
Alex Goodman, Anchore
Charles Hart, Hitachi Ltd.
Paul Horton, Sonatype Inc.
Nisha Kumar, Oracle
Bob Martin, MITRE
Katherine McAdams, HPE Aruba
Deanna Medina, Honeywell
Jeff Miller, NowSecure
Samuel Moore, T-Mobile
Behzad Mottahed, BD
Christine O'Leary, Intel
Surendra Pathak, Interlynk Inc.
Bill Pelletier, ZOLL Medical
Dmitry Raidman, Cybeats
Jim Routh, Retired
Aditi Sharma, Dell Technologies
David Shelly, GE
John Schiel, Lumen Technologies
Rich Steenwyk, GE HealthCare
Jeremiah Stoddard, INL
Allan Friedman, CISA
Megan Doscher, CISA
Justin Murphy, CISA
Kuldeep Sandhu, CISA
Jonathan Spring, CISA
Hendrik Tjoelker, Hanze University Groningen
Oscar van der Meer, MergeBase
Bhargav Vivekanandan, Blue Shield of California
David A. Wheeler, Linux Foundation
Jeff Williams, Contrast Security
Janet Worthington, Forrester Research

Curtis Yanko, GrammaTech
Keith Zantow, Anchore

Others participated, but do not wish to be named. Input into this document was also provided by some members of the OpenSSF “SBOM Everywhere” SIG and as well as some members of the CISA SBOM “Tooling and Implementation” workstream.