Carnegie Mellon University
Software Engineering Institute

Software Bill of Materials (SBOM) Harmonization Plugfest 2024

David Tobar
Jessie Jamieson
Mark Priest
Sasank Vishnubhatla
Jason Fricke

July 2025

SPECIAL REPORT
CMU/SEI-2025-SR-002
DOI: 10.1184/R1/28893080

CERT® Division

https://www.sei.cmu.edu

Table of Contents

List of Figures

List of Tables

Executive Summary

In this report, we at the Software Engineering Institute (SEI) describe the research findings and recommendations that resulted from SBOM Harmonization Plugfest 2024. The SEI organized and managed the Plugfest and conducted research into software bills of material (SBOMs) in support of the Cybersecurity and Infrastructure Security Agency (CISA).

The SEI's SBOM research included analyzing the differences among SBOMs and identifying the root causes of those differences.

We held the Plugfest to help vendors, standards producers, and the SBOM community understand how differences in how SBOMs are generated can result in different SBOM outputs. By gaining a better understanding of what causes these differences, we hope to recommend ways to ensure more predictable and higher quality SBOMs.

The SEI hosted a public meeting on November 19, 2024 to kick off the Plugfest. The Plugfest's SBOM submission phase—when we accepted SBOMs from prospective participants—lasted until December 15, 2024. This phase was followed by the research phase, which lasted through March 2025.

This report contains six major sections:

- **Section 1**: We introduce the Plugfest project, including background about the Plugfest process and SBOMs in general.
- **Section 2**: We provide a more detailed explanation of how we managed the Plugfest, the goals we established for it, our methodology, the analysis tools we used and developed, and the criteria we used to evaluate the SBOMs. We also provide a short description of the SBOMs we generated to establish a baseline for comparison.
- **Section 3**: We provide an overview of the SBOMs submitted by Plugfest participants.
- **Section 4**: We provide overall metrics that represent the depth and structure of the SBOMs we received. (We provide in-depth reviews of the metrics for each software target in the Appendix.)
- **Section 5**: We provide our findings from this research effort.
- **Section 6**: We provide our recommendations based on lessons learned. These recommendations include those related to SBOM harmonization, future research, and future SBOM Plugfests.

We received 243 SBOMs from the 21 Plugfest participants, which covered the nine Plugfest software targets. Notable findings from the Plugfest include the following:

1. We found significant variance in both the number of components and the content of the minimum required elements in SBOMs from different participants for the same software at the same lifecycle phase.
2. We found that some variance in SBOM content is due to the lack of normalization; the same content was simply being written differently (e.g., software version detailed as *v 2.0* or just *2.0*).

3. We discovered that some variance in SBOM content is due to differences in whether participants included minimum elements or not, which may be due to the somewhat artificial nature of generating SBOMs for a research project.

4. The wide variety of SBOM use cases may also be responsible for the lack of harmonization across SBOMs, even for those generated for the same target. Perhaps if we had specified purposes for each use case, participants may have taken a more harmonized approach to how they generated, enriched, and/or augmented their SBOMs for that use case.

   Some participants interpreted the meaning of the term *dependency* differently than others, and those differences affected what they included in the SBOM. Some participants' SBOM submissions included dependencies of first-party components that are not typically deployed, such as target documentation build tools, Continuous Integration and Continuous Deployment (CI/CD) pipeline components, and optional language bindings. We found that some differences in submitted SBOMs were because participants targeted different use cases, not necessarily because a tool was unable to discover dependencies. The variance in the depth of SBOMs for the same target also indicates that participants' expectations varied about the levels of transparency their SBOM should provide.

5. Participants used different approaches to generate their Build SBOMs, which led to differences in the components discovered. Some participants used a container build process to generate their Build SBOM, and others built a standalone executable for their chosen runtime environment using the target's language or build-framework-specific process. Build SBOMs also varied based on the environment and tool configurations each participant used.

6. In some cases, participants used different approaches to generate their Source SBOMs. Source SBOMs capture dependencies declared or inferred from source code. Some participants used additional information from external locations, such as the artifact repositories referenced by dependencies or the contents of platform toolchain libraries to infer additional dependencies.

See Section 5 for details about our findings and Section 6 for details about our recommendations.

To enable further research, the SEI is hosting a repository of SBOMs submitted by Plugfest participants who agreed to make their SBOMs public.[1] We expect lessons learned from the Plugfest will be useful to SBOM vendors, standards producers, and the SBOM community.

[1] The SBOM Plugfest 2024 repository is being hosted on GitHub at https://github.com/cmu-sei/sbom-plugfest-2024.

Abstract

This report describes the research findings and recommendations that resulted from the 2024 SBOM Harmonization Plugfest research project. The Software Engineering Institute (SEI) project team managed the Plugfest and conducted research into the submitted software bills of material (SBOMs) in support of Cybersecurity and Infrastructure Security Agency (CISA). In this project, the SEI focused on understanding how differences in SBOM generation can result in different SBOM outputs. After gaining a better understanding of what causes these differences, the SEI project team developed recommendations for organizations to ensure more predictable and higher quality SBOMs. This report contains six major sections: an introduction, an explanation of the SBOM Plugfest process, an overview of SBOM submissions from participants, a description of the SEI project team's analysis, the team's findings, and the team's recommendations.

1 Introduction

We are Software Engineering Institute (SEI) researchers studying software bills of material (SBOMs). Our goal for this research is to support the harmonization of SBOM implementation. In this report, we describe our findings and recommendations that resulted from SBOM Harmonization Plugfest 2024.

1.1 Task

The SEI organized and managed the Plugfest to support the Cybersecurity and Infrastructure Security Agency (CISA), which sponsored the SEI's SBOM research. A plugfest event allows participants to demonstrate and test the interoperability of their tools to continually meet evolving technical standards.

We designed the Plugfest to help vendors, standards producers, and the SBOM community understand how differences in SBOM generation can result in different SBOM outputs. Analyzing a piece of software at the same point in its lifecycle should produce similar SBOMs. However, it is common for different SBOM tools to generate divergent SBOM results, which can undermine the confidence users have in SBOMs. We did not intend the Plugfest to be a competition or "bake-off" between SBOM producers. Rather, we intended this Plugfest to enable us to evaluate how much variance or commonality in SBOMs occurs for the same software when different participants (e.g., industry, private sector, academia) use different tools to generate them.

We officially announced the Plugfest on the SEI's social media channels on November 8, 2024. We subsequently held a public kickoff meeting on November 19, 2024, to provide additional details about the Plugfest and answer questions from interested parties. We gave interested parties until December 15, 2024, to elect to participate in the Plugfest by submitting at least two SBOMs. We then performed the following:

- analyzed the submitted SBOMs
- outbriefed the participants
- produced a report for CISA that described our Plugfest research

We designed the Plugfest to help SBOM practitioners (e.g., producers, consumers) learn about what causes differences in SBOMs for the same software target at the same stage of the software lifecycle (e.g., Source, Build).[2] Once we better understood what causes these differences, we formed recommendations to help the community generate more predictable and higher quality SBOMs. (See Section 6 for details.)

[2] CISA describes various SBOM types, including Build and Source, in *Types of Software Bill of Material (SBOM) Documents* [CISA 2023].

Our SBOM research includes analyzing the differences that appear among SBOMs and the root causes of those differences. These root causes include imprecise definitions or standards, how uncertainty is addressed, and other implementation decisions.

1.2 Background

An SBOM is "a formal, machine-readable inventory of software Components and Dependencies, information about those Components, and their relationships. An SBOM's inventory should be as comprehensive as possible and should explicitly state where relationships cannot be articulated" [CISA 2024].

1.2.1 SBOM Background

The predecessor of the SBOM—the bill of materials (BOM)—was first used in World War I, when the scarcity of materials led to more efficient methods of managing materials. While BOMs can include software components, the SBOM specifically focuses on software, including identifying libraries, dependencies, and versions.

In 2019, the U.S. National Telecommunications and Information Administration (NTIA) published the first version of *Framing Software Component Transparency: Establishing a Common Software Bill of Materials (SBOM)* [CISA 2024]. This publication was a product of the NTIA Multistakeholder Process on Software Component Transparency Framing Working Group. That group's stated goal was creating "a model for software component information that can be universally and transparently shared across industry sectors" [NTIA 2024a]. The second version of this publication, released in 2021, added required SBOM attributes and CycloneDX as an SBOM format.

The Department of Commerce (DOC) published *The Minimum Elements for a Software Bill of Materials (SBOM)* on July 12, 2021 [DOC 2021].[3] This DOC publication defines the minimum elements for an SBOM. These minimum elements address three base use cases: vulnerability management, software inventory, and software licenses. The three broad categories of minimum elements that support these use cases are data fields, automation support, and practices and processes.

NTIA identifies three key SBOM data exchange formats: Software Package Data eXchange (SPDX®), CycloneDX, and National Institute of Standards and Technology's (NIST's) Software Identification (SWID) Tagging [NTIA 2021a]. (Only SPDX and CycloneDX are considered complete SBOM formats.)

The Linux Foundation created SPDX, a project it hosted as part of its Open Source Compliance Program as a data exchange format to enable information sharing about software packages. This format is now *ISO/IEC 5962:2021 Information Technology—SPDX® Specification*, an internationally recognized data format standard for communicating the component and metadata information associated with software packages [ISO/IEC 2021]. SPDX inventories software

[3] Executive Order 14028, *Improving the Nation's Cybersecurity*, directed the DOC, in coordination with NTIA, to publish the minimum elements for an SBOM [White House 2021]

components, license and copyright information, and security references. The current Linux Foundation version of SPDX is 3.0.

The Open Worldwide Application Security Project (OWASP) Foundation created CycloneDX to be used with OWASP Dependency-Track, "an intelligent component analysis platform that allows organizations to identify and reduce risk in the software supply chain" [OWASP 2025a, 2025b]. CycloneDX v1.6 has been ratified as an Ecma International standard. CycloneDX may be used as a global xBOM standard across multiple domains, including software, services, hardware, firmware, artificial intelligence, machine learning, and cryptography [OWASP 2024].

1.2.2 SBOM Plugfest Background

This SBOM Plugfest followed in the footsteps of previous SBOM plugfests. Those plugfests also relied on participation from volunteers who contributed their "sweat equity" by submitting SBOMs for specified software targets. Volunteers in NTIA's Software Component Transparency initiative facilitated the first SBOM Plugfest, which was held on April 9, 2021. It focused on SBOM generation and consumption. At that Plugfest, organizers selected four software targets, and 13 organizations submitted SBOMs.[4]

Volunteers from the Organization for the Advancement of Structured Information Standards (OASIS) managed the second Plugfest, which was held on June 22, 2021. At the second Plugfest, organizers expanded on the set of four software targets from the initial Plugfest with an additional five targets.[5]

[4] For more information about the first SBOM Plugfest, see the SBOM Plugfest I Summary on Google Docs [NTIA 2025].

[5] For more information about the second Plugfest, see the Plugfest #2 information on Google Drive [OASIS 2021].

2 The Plugfest Process

In this section, we describe how we managed the Plugfest, the detailed instructions we provided to participants submitting SBOMs, and the methodology we used to extract information from the SBOMs, including SBOM analysis tools, software target dependency inspection, evaluation criteria, the baseline SBOMs we used for comparison, and the calculations of the depth and breadth of the SBOMs we received.

2.1 About the Plugfest

As part of managing the Plugfest, we asked the SBOM community to contribute SBOMs that they generated based on specified software targets. We selected seven software targets (i.e., the first seven targets in Table 1) as an initial representative sample of various programming languages and processes. We deliberately selected these software targets to explore trends in SBOM agreement and divergence in correlation with various attributes of software packages and libraries. Based on community feedback, we added two more software targets—PHPMailer and jq (representing PHP and C)—for a total of nine final targets. See Table 1 for details about the nine specified targets.

We provided directions to the SBOM community that specified the exact software package and package version that participants would use to develop SBOMs for each target. These specifications ensured that all participants based their SBOMs on the same sources.

We conducted an initial virtual meeting with interested parties to review Plugfest processes, directions, and expectations. We approved to participate in the Plugfest any interested parties who invested their time and effort using their SBOM tools to generate and submit at least two SBOMs for any of the nine software targets.

We gave volunteer participants until December 15, 2024, to submit SBOMs for the target software. We asked these participants to generate Build and/or Source SBOMs in standard data formats (SPDX or CycloneDX). In late January 2025, we held a session to review our initial analysis results with Plugfest participants.

At the end of the Plugfest research project, we asked Plugfest participants to approve making their SBOM submissions public. Those SBOMs are now available for further research on the SEI's GitHub® site.[6]

[6] The SBOM Plugfest 2024 repository is being hosted on GitHub at https://github.com/cmu-sei/sbom-plugfest-2024.

Table 1: Software Targets

Software	Version (GitHub)	Language	For More Information (GitHub)	What does the project do?
NodeJS-goof	Commit d240896 2023	JavaScript	Snyk Labs	A vulnerable Node.js demo application based on the Dreamers Lab tutorial
HTTPie	Commit f4cf43e July 2024	Python	HTTPie cli	HTTPie for Terminal
MineColonies	Commit 7c184da·Oct 2024	Java	minecolonies	An interactive building mod that allows you to create a town within Minecraft
OpenCV	Commit 3919f33 17 Oct, 2024	C++	OpenCV	An Open-Source Computer Vision Library
Gin	Commit: f05f966 Sept 2024	Go	Gin-Gonic	An HTTP web framework written in Go (Golang)
Hexyl	Commit 427a552 Sept 2024	Rust	sharkdp	A command-line hex viewer
Dependency Track	4.12.1 - 25 Oct, 2024	OCI - Java[7]	Dependency-Track	An intelligent Component Analysis platform that allows organizations to identify and reduce risk in the software supply chain and leverages SBOMs to provide capabilities that traditional Software Composition Analysis (SCA) solutions cannot achieve
PHPMailer	Commit 182f7b9 · 15 Oct, 2024	PHP	PHPMailer	The classic email-sending library for PHP that is a full-featured email creation and transfer class for PHP
jq	Commit 96e8d89·20 Nov, 2024	C	jqlang/jq	A lightweight and flexible command-line JSON processor

[7] OCI stands for Oracle Cloud Infrastructure.

2.2 Submission Instructions

Table 2 details the instructions we provided to participants on how to submit SBOMs, which we published on an SEI webpage dedicated to the Plugfest.

Table 2: Submission Instructions for the Plugfest

#	Instruction
1	Full participation is open to anyone who submits SBOMs for at least two of the eight software targets. The submission deadline is December 15, 2024.
2	Create a folder for your organization (or tool) in the SBOM Plugfest 2024 directory. If you wish, you may secure it so that only you and the CISA/SEI analysts have read access.
3	Store your SBOM results using the following directory structure: <organization>/<target name>/<file format>.
4	Submit (Source, Build) SBOMs in either or both standards. Use the following file-naming conventions for the SBOMs: a. SPDX: example. → example.spdx.json or example.spdx or example.spdx.xml (For more information, see the SPDX website.) b. CycloneDX: example/cyclonedx/bom.xml → example.cyclonedx.bom.xml
5	Enrich the SBOM as you normally would.
6	Validate your SBOM before submitting it and consider using one of the following tools: a. SPDX: https://tools.spdx.org/app/ntia_checker, https://tools.spdx.org/app/validate b. CycloneDX: https://github.com/CycloneDX/sbom-utility
7	Upload a README file that provides orientation and context for reviewers and includes the following information: a. point of contact (POC) for the SBOM submission b. version of the tool being used c. types of SBOMs being represented (e.g., Source, Build) d. how the SBOM was validated, including the name of the tool used e. additional information that might be useful to reviewers (e.g., details on any manual edits or enrichments made to the tool-generated SBOM)
8	Add the SBOM files generated for the reference examples to your tool's folder.

2.3 Methodology

We used an analytic methodology for this Plugfest that comprised a combination of automated and manual processes for extracting data from the SBOMs. Our approach to evaluating the SBOMs included doing the following:

- conducting a quantitative review using tools that processed and reported on the content of the SBOMs
- conducting a qualitative review by having subject matter experts (SMEs) review the SBOMs

We provide more details in Sections 2.3.1–2.3.4 about the SBOM analysis tools we developed to facilitate reviewing and analyzing the SBOMs, the software target dependency inspection process we used to understand expected dependencies, the evaluation criteria we used in our analysis, the baseline SBOMs we generated, and the depth and breadth of the SBOMs we analyzed. The Appendix further describes the detailed results of our analysis for each software target, including the baseline SBOMs we generated for comparison purposes.

2.3.1 SBOM Analysis Tools

We developed software tools to facilitate reviewing and analyzing the SBOMs that Plugfest participants submitted. Due to the many submissions we received for the Plugfest, we focused our tool development on automating the ingesting and processing of SBOMs to collect, collate, and export data about each one.

Participants submitted SBOMs in SPDX and CycloneDX formats in a variety of encodings, including JSON, XML, and YML.[8] Due to the potential differences between SPDX and CycloneDX SBOM formats, our initial analysis grouped the two formats separately. Since the majority of participants encoded their output in JSON, we prioritized JSON SBOMs for analysis. We also decided to assess SBOMs in other formats (e.g., XML, YML) based on the time and resources available. We wrote code for processing SBOMs using Python within Jupyter computational notebooks hosted on an SEI internal Bitbucket® repository, which also contained a copy of SBOM Plugfest submissions. We chose this software development methodology primarily to facilitate a quick-turn, accurate, and collaborative exploratory analysis.

We used two primary notebooks for analyzing SBOM submissions: one for CycloneDX and one for SPDX. We sought to extract the following from each SBOM:

- the fundamental information related to the presence or absence of minimum elements
- information about software components, including their relationships to one another and with the target software.[9]

In each notebook, we collected information from each SBOM by doing the following:

- traversing the directory of SBOM submissions, importing JSON SBOM files, and decoding the JSON files so that data could be extracted
- extracting minimum elements from each SBOM where the data existed and noting where data was missing
- constructing a dependency tree based on the dependencies listed in each SBOM (These dependency trees contained information about software components and the types of relationships among those components as listed in the SBOM.)
- collating data from each SBOM into two common data structures: one for information related to minimum elements and the other for component information. (We also tagged data extracted from each SBOM with whether the SBOM was a Build or Source SBOM and which target the SBOM applied to. For traceability and validation purposes, we also included the file path for the data.)

We then analyzed the data structures using Python data science packages, or we exported them as comma separated value (CSV) files for further analysis using other tools. We used information about the presence or absence of minimum elements to generate summary statistics for each

[8] JSON stands for JavaScript Object Notation, XML stands for eXtensible Markup Language, and YML is the file format produced by YAML, which stands for YAML Ain't Markup Language, but originally stood for Yet Another Markup Language.

[9] For more information about minimum elements for SBOMs, refer to Section 2.3.2.

software target and each SBOM type (Source/Build). Meanwhile, we used dependency graph information to analyze the presence/absence of components and assess the depth of the SBOMs. (For more information about SBOM depth, refer to Section 4.)

We are currently evaluating the code we developed for this Plugfest to prepare it for release in an open source repository of Python packages and scripts. A release date will be determined pending further analysis.

2.3.2 Software Target Dependency Inspection

Many open source software projects follow programming language and platform-specific conventions for declaring third-party dependencies and defining a repeatable build process. These conventions usually allow for specifying a single version or a range of acceptable versions for the declared dependencies. Software projects commonly use artifact repositories with associated package managers to obtain their dependencies rather than building them from source code.

We manually inspected each target to determine its declared dependencies and captured this information in dependency tables we provide for each software target in the Appendix. When a target followed an identified convention, we extracted the declared dependencies and allowable versions from their associated artifacts and analyzed the submitted SBOMs to detect the presence or absence of these dependencies.

We used a permissive approach for dependency name and version matching because the conventions for these values vary widely across artifact repositories used to host dependencies and among current SBOM tools. We considered an SBOM to include a dependency if the SBOM output clearly showed it did, even if the reported name did not exactly match the name specified in the target. Similarly, we considered a dependency to be present regardless of whether the version listed in the SBOM was an allowable version based on the target's dependency artifacts.

We intended this manual inspection to determine whether a dependency was discovered rather than to adjudicate the validity of SBOM field values. In our analysis, we considered only the dependency artifacts for the first-party components that are clearly intended for runtime deployment whenever the target is used as part of a software solution. Several targets included multiple artifacts intended for building project documentation, language bindings for other supported programming languages, and other secondary functions. We did not include these artifacts in the analysis, but they were analyzed and used by some baseline tools and tools used to generate participant submissions. Not including these artifacts accounts for some components that were present in SBOMs but did not explicitly declare as third-party dependencies.

2.3.3 Evaluation Criteria

To better understand sources of divergence for SBOMs, we sought to establish criteria we could use to evaluate SBOM alignment. We chose criteria that were in line with NTIA recommendations about the quality attributes of SBOMs in its *Roles and Benefits for SBOM Across the Supply Chain* [NTIA 2019].

We selected these criteria because of their relevance to use cases for SBOMs within the cybersecurity community. Where possible, we developed metrics that we could use to understand the variance within criteria across SBOM submissions and types for each software target and phase.

Table 3: Evaluation Criteria for SBOMs

Criteria	Definition	Evaluation	Metric(s)
Completeness	Information about all the components that make up a piece of software and the process(es) used to assemble them	Determine whether SBOMs contain information about the target software package, its dependencies, and its requirements as stated by the producers of the target software packages, and whether that information is correct.	• Depth • Breadth[10] • Minimum elements • Component information (e.g., version numbers, license information)
Accuracy	A determination of whether information about all the components that make up a piece of software and the process(es) used to assemble them are accurate		
Pedigree	The term of art for having information on all of the components that have come together to make a piece of software and the process used to assemble them		
Provenance	The term of art for having information about the chain of custody of the software and all of the components that comprise that software, capturing information about the authors and locations where the components were obtained from	Determine the presence of accurate SBOM authorship information.	• Authorship information that is present and accurate
Integrity	The use of cryptographic techniques to indicate that (1) the SBOM has not been altered since the author initially wrote it or (2) if there was a modification that was made by a subsequent SBOM author	Determine the presence of a cryptographic hash for the SBOM and information about the technique used to generate the hash.	• Cryptographic hash that is present • Algorithm used to create the hash that is present (e.g. SHA-256, MD5)

The NTIA recommendations on SBOMs include certain data fields as a type of minimum element in any SBOM. Table 4 illustrates the minimum elements for SBOMs as well as a mapping of these elements to corresponding data values in SBOM standards.[11]

[10] Breadth, as we define it in this report, was considered as a possible metric by which SBOM consistency could be measured; however, ultimately, we did not focus on this metric as it was clear that breadth varied across submitted SBOMs. Although it was calculated for SBOMs, we did not report this value during our report of findings. We include it here for completeness.

[11] Table 4 is excerpted from Table 1 in *Framing Software Component Transparency: Establishing a Common Software Bill of Materials (SBOM)* [CISA 2024].

Assessing the consistency of the minimum elements of SBOMs submitted as part of the Plugfest was a central part of our analytic methodology and was a critical component in determining their completeness and accuracy. The minimum elements listed below can apply to both the target software package and the component software packages listed as dependencies within SBOMs. For this Plugfest, we derived summary statistics and evaluations of consistency across SBOMs with respect to minimum elements from those elements that correspond to the target software package.

Table 4: Minimum SBOM Elements Mapped to Existing Formats

Element	SPDX 3.0[12]	CycloneDX v1.6 (ECMA-424)
SBOM Author Name	Core.CreationInfo.createdBy	metadata.authors
SBOM Timestamp	Core.CreationInfo.created	metadata.timestamp
SBOM Type	Software.Sbom.sbomType	metadata.lifecycles
SBOM Primary Component	Software.Sbom.rootElement	metadata.component
Component Name	Software.Package.name	components[].name
Component Version String	Software.Package.packageVersion	components[].version
Component Supplier Name	Software.Package.suppliedBy	metadata.supplier components[].supplier
Component Cryptographic Hash	Software.Package.verifiedUsing	components[].hashes[]
Component Unique Identifier	Core.Artifact.spdxId Software.SoftwareArtifact.contentIdentifier Software.SoftwareArtifact.externalIdentifier (cpe22, cpe23, cve, gitoid, packageUrl, swhid, swid, securityOther, other)	serialNumber + version components[].cpe components[].purl components[].swid components[].omniborId components[].swhid components[].evidence.identity
Component Relationships	Core.Relationship Contains dependsOn hasStaticLink hasDynamicLink hasProvidedDependency hasOptionalDependency	dependencies[] components[].components

[12] We use SPDX V3.0 in this table since it is the current version reflected in the CISA reference, although the great majority of the SBOMs we received were in SPDX V2.3.

Element	SPDX 3.0[12]	CycloneDX v1.6 (ECMA-424)
Component License	Core.Relationship hasConcludedLicense hasDeclaredLicense	components[].licenses[] components[].licenses[].acknowl-edgement[declared, concluded] components[].licenses[].licensing (proprietary) components[].evidence.licenses[]
Component Copyright Holder	Software.SoftwareArtifact.copy-rightText	components[].copyright components[].evidence.copyright

2.3.4 Baseline SBOMs

We generated Source SBOMs for each software target and standard to serve as examples of what we might expect to see in SBOMs submitted by Plugfest participants. These examples allowed us to start building our analysis tools before submissions were received. We selected Syft, Trivy, and Microsoft® SBOM tool to create these baseline SBOMs based on our previous work in this area. In that work, we found that these open source, community-developed tools provide an acceptable depth and breadth in their SBOMs' coverage of software components. We used Syft and Trivy to create both CycloneDX and SPDX Source SBOMs. We used Microsoft SBOM tool to create SPDX Source SBOMs. We built the baseline SBOMs from locally cloned copies of the target Git repositories at the specified commit hashes.

The baseline SBOMs proved useful in our analysis because they helped us understand some of the reasons for the differences that we saw across tool-generated SBOMs for the same target. Because the tools were open source and we ran them with known settings in a controlled environment, we could reason more effectively about the differences that we found among them. By inspecting the tools' logs and source code, we were able to determine why one tool discovered a component that another did not, for example. The baseline SBOMs set our expectations for what SBOM providers could generate by simply using available open source SBOM tools. The Appendix includes summary information about the baseline SBOMs for each target. We generated baseline commonality charts from the SPDX results since all three tools support that format. We did not observe meaningful differences for any tool based on format alone.

2.3.5 SBOM Depth and Breadth

For this Plugfest, we defined the *depth* of an SBOM to be the length of the longest path in the directed graph generated by the dependency tree defined by the components and relationships within the SBOM. Similarly, we calculated *breadth* as the maximum number of components at any given distance from the target software in the dependency tree defined by the components and relationships within the SBOM.

Figure 1 illustrates an example of a *dependency tree*. In this example, the depth is three, since the longest path in this directed graph is three. The breadth is three, since the largest number of components at any given distance from the target is the set of three direct dependencies. Figure 1 illustrates the depth and breadth by the rounded rectangle and square rectangle, respectively.

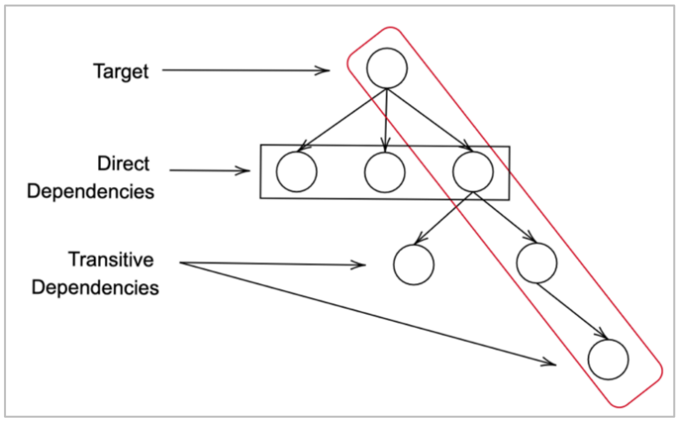

Figure 1: Example Dependency Tree Illustrating Target Software Dependencies, Depth, and Breadth

As described in the NTIA's *The Minimum Elements for a Software Bill of Materials (SBOM),* SBOMs should contain "all primary (top level) components, with all their transitive dependencies listed" [DOC 2021]. SBOMs that contain top-level dependencies should contain enough detail to allow for transitive dependencies to be identified recursively. Depth in the dependency tree of software provides transparency into components and subcomponents of the software. SBOMs that are shallow may require additional augmentation with component SBOMs to reach the desired level of transparency.

A variety of relationship types may define dependencies within SBOMs. For example, in the CycloneDX standard for SBOMs, dependencies "represent the relationships between components or services that a given component relies on functionally, focusing exclusively on the connections rather than the inventory of components," and these relationships make use of the "dependsOn" dependency type within the format specification [OWASP 2025c]. Likewise, the SPDX specification provides for a number of dependency types to be specified within the relationship fields of an SBOM. Because of the aforementioned variety of relationships possible within an SBOM and the variety of potential use cases for SBOMs, the depth of an SBOM required to provide a sufficient level of transparency for its use case may vary. For more details about the specific technique we used to calculate depth in SBOMs for the Plugfest, see Section 4.

3 Summary of SBOM Submissions

In this section, we provide an overview of the SBOM submissions from Plugfest participants. See the Appendix for details of our analyses of nine different software target JSON SBOMs. Detailed analysis findings from the Plugfest are included in Section 5.

Figure 2 shows the distribution of SBOMs we received from Plugfest participants across the various software targets. There were 21 Plugfest participants, and each was asked to submit at least two SBOMs to participate. Of those 21 participants, five submitted over 10 SBOMs. As a result, we received a total of 243 SBOMs.

To ensure participants' anonymity and to prevent any bias in our review, we anonymized participant names by assigning alphanumeric codes to each. One participant, who was assigned the code Y2, submitted many more SBOMs (102) than all the others.

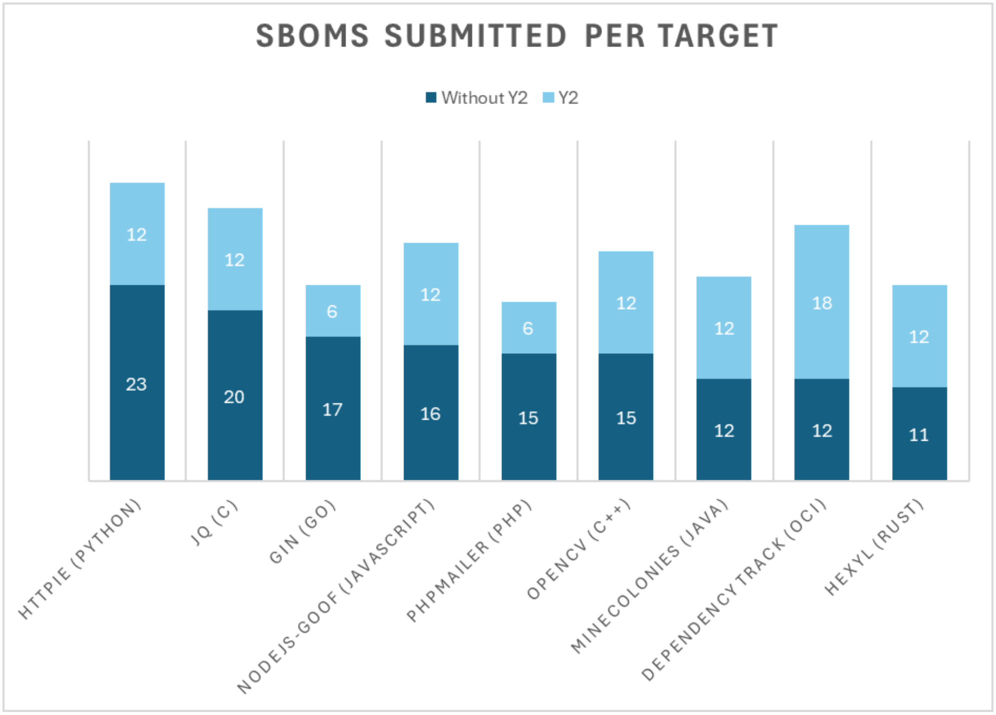

Figure 2: SBOMs Submitted per Target

Figure 3 shows the distribution of SBOMs across SBOM standards and formats. Once we received the submissions, we determined that our analysis of SBOMs would proceed in the following order:

- the most common SBOMs: JSON SBOMs in CycloneDX format
- JSON SBOMs in SPDX format
- XML and YML SBOMs, which were treated as special cases

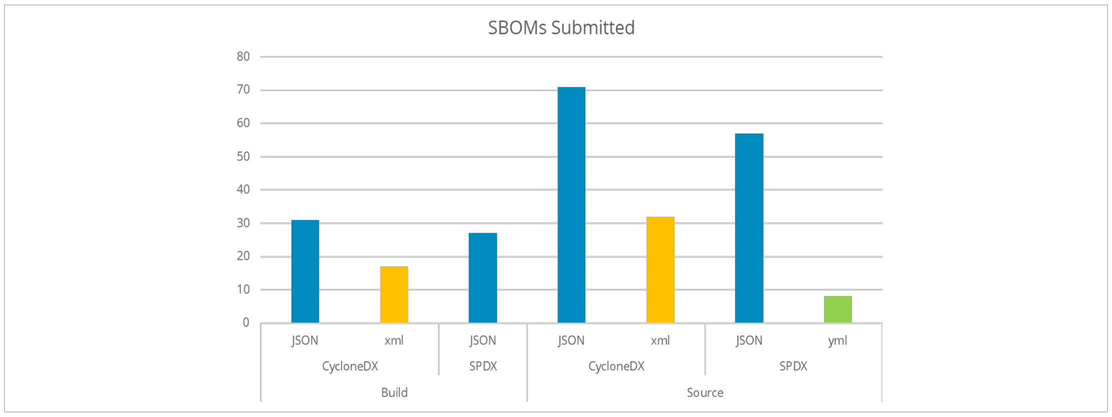

Figure 3: Distribution of Submitted SBOMs by Standard and Format

Table 5 shows the distribution of JSON SBOMs across the various software targets for both Build and Source SBOMs.

Table 5: Distribution of JSON SBOMs

SBOM/Type	CycloneDX SBOMs	SPDX SBOMs	Grand Total
Build	**31**	**27**	**58**
Dependency Track (OCI)	5	5	10
Gin (Go)	2	1	3
Hexyl (Rust)	3	4	7
HTTPie (Python)	4	5	9
jq (C)	5	4	9
MineColonies (Java)	4	2	6
NodeJS-goof (JavaScript)	4	3	7
OpenCV (C++)	4	3	7
Source	**71**	**57**	**128**
Dependency Track (OCI)	6	6	12
Gin (Go)	8	7	15
Hexyl (Rust)	6	4	10
HTTPie (Python)	11	10	21
jq (C)	9	8	17
MineColonies (Java)	6	5	11
NodeJS-goof (JavaScript)	8	5	13
OpenCV (C++)	8	5	13
PHPMailer (PHP)	9	7	16
Grand Total	**102**	**84**	**186**

We received a limited number of XML and YML SBOM submissions. Table 6 shows the distribution of these SBOMs across the various software targets for both Build and Source SBOMs. Only three participants contributed to the 49 XML SBOMs, and of those, only one participant contributed 34 XML SBOMs. Only one participant contributed all eight YML SBOMs. Because we had only one YML submission per target, we could not compare these SBOMs with other

YML SBOMs for the same target. Due to the lack of time, we were unable to thoroughly review the XML and YML SBOMs.

Table 6: Distribution of XML and YML SBOMs

SBOM/Type	Count of XML SBOMs	Count of YML SBOMs
Build	**17**	**0**
Dependency Track (OCI)	4	0
Hexyl (Rust)	2	0
HTTPie (Python)	2	0
Jq (C)	2	0
MineColonies (Java)	2	0
NodeJS-goof (JavaScript)	3	0
OpenCV (C++)	2	0
Source	**32**	**8**
Dependency Track (OCI)	3	1
Gin (Go)	4	1
Hexyl (Rust)	5	1
HTTPie (Python)	3	0
jq (C)	3	1
MineColonies (Java)	4	1
NodeJS-goof (JavaScript)	4	1
OpenCV (C++)	2	1
PHPMailer (PHP)	4	1
Grand Total	**49**	**8**

4 SBOM Depth Analysis

In this section, we provide our analysis of the depth of various SBOMs for the software targets.

Table 7 presents the depths of SBOM submissions and baseline SBOMs. For each target, format, and lifecycle phase, we calculated the maximum depth of SBOMs in each category as well as the median depth for SBOMs in each category. We also calculated maximum and median depths for the combination of formats for each target/lifecycle phase.

We calculated depth values for SPDX v2.3 SBOMs by generating a dependency tree from "DEPENDS ON" and "DEPENDENCY_OF" relationships in each SBOM. Likewise, we calculated depth values for CycloneDX format SBOMs by generating a dependency tree from the "DEPENDENCIES" elements listed in each SBOM. We chose this approach primarily to enable the comparison of depth calculations for both specifications of SBOMs. However, an analysis of relationship types for SPDX SBOMs that had their depths calculated revealed that the only relationship types listed in these SBOMs were "METAFILE_OF," "CONTAINS," "DESCRIBES," "DEPENDS_ON," "DEPENDENCY_OF," and "OTHER" despite many other relationship types being allowed within the SPDX standard.

Depths of the submitted SBOMs ranged widely, and some SBOMs did not include dependency information or information about transitive dependencies. Studying the overall structure of SBOMs is challenging. Additional research time would help in understanding SBOM structures and dependencies as well as the root causes of discrepancies across SBOMs generated for the same software target.

Table 7: Depth Calculations for Submitted SBOMs and Baseline SBOMs, by Type, Target, and Format

		CycloneDX Submissions			**SPDX Submissions**			**SPDX and CycloneDX Submissions**			**Baseline SBOMs**		
	Type/Target	**Max. Depth**	**Median Depth**	**Number of SBOMs**	**Max. Depth**	**Median Depth**	**Number of SBOMs**	**Max. Depth**	**Median Depth**	**Number of SBOMs**	**Max. Depth**	**Median Depth**	**Number of SBOMs**
Build	Dependency Track	10	10	3	4	3	3	10	3	6	N/A		
	Gin	2	1	2	-	-	-	2	1	2			
	Hexyl	10	10	3	4	4	3	10	6.5	6			
	HTTPie	5	3	4	3	2	4	4	2.5	8			
	jq	2	1	5	2	2	2	2	1	7			
	MineColonies	12	1	4	2	2	2	12	1	6			
	NodeJS-goof	18	17.5	2	8	8	3	18	17.5	5			
	OpenCV	2	2	4	2	2	2	2	1.5	6			
Source	Dependency Track	10	2.5	6	6	1	6	10	2	12	8	1	5
	Gin	3	1	7	3	1	4	3	1	11	3	1	5
	Hexyl	10	6	6	5	4	4	10	4	10	10	8	5
	HTTPie	6	2	8	16	1	5	16	1	13	4	0	5
	jq	3	2	8	23	1	4	23	1	12	2	2	5
	MineColonies	5	2	6	2	1	4	5	1	10	0	0	5
	NodeJS-goof	17	2.5	8	9	1	5	17	2	13	17	8	5
	OpenCV	3	2	6	12	1	3	12	2	9	3	1	5
	PHPMailer	5	2	9	3	1	5	5	1.5	14	0	0	5

5 Findings

This section describes the key findings of our analysis of the SBOMs submitted for the Plugfest effort.

- **Commonality and Variance in SBOM Components.** Our review found significant variance in the number of components in SBOMs from different participants for the same software at the same lifecycle phase. The variance is visible in the commonality charts provided in the Appendix. For example, Figure 9 includes the chart for the SPDX Source SBOMS for NodeJS-goof, where five SBOMs included 496 components, but one SBOM included 1081 components. Based solely on the SBOMs we received, we found that the software targets with the highest commonality in components were Hexyl (Rust) and NodeJS (JavaScript). We found that jq (C) and OpenCV (C++) were the targets with the lowest commonality in components. Figure 22 includes the chart for the SPDX Source SBOMS for jq, where three SBOMs included under 30 components, but one SBOM included 745 components.
- **Component Name and Version Challenges.** Our review showed that the lack of normalization for component names and versions (e.g., software version detailed as *v 2.0* or just *2.0*) is also a cause of variance in SBOMs [MITRE 2024].
- **Version Ranges and Specification.** When a target allows for version flexibility, there is ambiguity about what versions should be included in a source SBOM because SBOM specifications allow only a single version to be normatively captured for each dependency. One participant stated in a readme file that they deliberately chose the minimum allowable version for each dependency as a kind of worst-case analysis. In general, multiple versions were listed across different participants for each target with version ranges. In a Build SBOM, a single version is always captured during a given build, but that version may differ across SBOMs because it is based on the specific build process, environment, and time that a build occurs. We note here that the Package Uniform Reference Locator (PURL) specification has a new syntax for version ranges called the *version range specifier.*[13]
- **Ease of Review.** It is easier to inspect SBOMs for targets that explicitly declare dependencies and follow common build conventions (e.g., HTTPie, NodeJS-goof, Hexyl).
- **Dependency Discovery Challenges.** Organizations use many conventions for specifying dependencies and building software across programming languages, platforms, environments, and software development frameworks. These differences make it challenging for any one SBOM tool or producer to capture and represent components and relationships for an arbitrary target. Our baseline SBOM analysis confirmed that this challenge explains at least some of the differences in discovered dependencies across tools.
- **Differing Definitions of Dependencies.** A review of submitted readme files and discussions with a few participants indicated that they had different definitions or interpretations of a

[13] For more information on the issues surrounding standardized formatting of version ranges and the new version range specifier syntax, see https://github.com/package-url/purl-spec/blob/main/VERSION-RANGE-SPEC.rst.

"dependency." A few submissions included dependencies of first-party components that are not typically deployed, such as target documentation build tools, CI/CD pipeline components, and optional language bindings. At least some of these differences are due to the SBOMs targeting different use cases and are not necessarily due to any tool's inability to discover dependencies. The variance in the depth of SBOMs for the same target also indicates participants' varying expectations of the levels of transparency provided by SBOMs.

- **Diversity of Use Cases.** SBOMs have diverse use cases, which lead to different types of SBOMs [NTIA 2019, CISA 2023]. In line with our findings about dependencies, the wide variety of use cases for SBOMs may be responsible for the expansion of SBOM specification as well as the lack of harmonization across SBOMs, even for those generated for the same target. Use cases for SBOMs may also vary by industry sector and risk model. Our guidance to Plugfest participants did not mandate a specific purpose in mind for the SBOMs other than general research. Discussions with participants indicated that if they had specific purposes in mind they may have generated, enriched, and/or augmented their SBOMs differently.
- **Build SBOM Variance.** PHPMailer, NodeJs-Goof, Dependency Track, HTTPie, and jq all include a Dockerfile and build artifacts that can be used to generate a Docker container as the build output. A few participants used this container build process to generate their Build SBOM, and others built a standalone executable for their chosen runtime environment using the target's language or build framework-specific process (e.g. npm, maven). These different approaches led to differences in the components they discovered. Build SBOMs also varied based on the environment and tool configurations each participant used. Discovering information about the build environments that the vendors used was possible in some cases by manually inspecting the SBOMs. In these instances, containers were listed as components in the SBOM. Participants also provided this information to us in readme files that explained their SBOM generation processes.
- **Source SBOM Variance.** Source SBOMs capture dependencies declared or inferred from source code. In a few cases, participants used additional information from external locations, such as the artifact repositories referenced by dependencies or the contents of platform toolchain libraries, to infer additional dependencies.
- **Minimum Elements.** We found significant variance in the degree of inclusion of the various minimum required elements in SBOMs from different participants for the same software at the same lifecycle phase. As shown in Table 8, some minimum elements were well populated (e.g., Target Name, Timestamp, Target Type), while others were not.

Table 8 shows the percentages of SBOMs that contained the given minimum elements for both the SBOMs submitted and the baseline SBOMs we generated for comparison analysis. In Table 8, we shaded cells in the Submitted SBOMs Overall column where the percentage was under 50% for the given SBOM element, indicating a significant lack of compliance for those minimum required elements.

Table 8: Review of Minimum Elements for Submitted and Baseline SBOMs

SBOM Element	Submitted SBOMs			Baseline SBOMs		
	CycloneDX	**SPDX**	**Overall**	**CycloneDX**	**SPDX**	**Overall**
Target Name	96%	100%	97%	100%	100%	100%
Timestamp	92%	100%	94%	100%	100%	100%
Target Version	50%	39%	44%	50%	67%	60%
SBOM Author	37%	32%	35%	0%	0%	0%
Cryptographic Hash	13%	33%	22%	0%	0%	0%
Lifecycle Phase	9%	None	5%	0%	None	0%
SBOM Supplier	9%	88%	42%	0%	100%	60%
License	3%	10%	6%	0%	67%	40%
Target Type	95%	81%	90%	100%	67%	80%

The overall percentage of SBOMs that included the Cryptographic Hash for their software target was only 22%, and none of the SBOMs for the same software target included the same hash.

Although most participants identified a target type for their SBOMs, there was a wide discrepancy in how they did so. For example, in the supplied SBOMs, jq was listed as an application, a container, data, and a file.

6 Recommendations

This section details our recommendations based on our analysis of the SBOMs submitted for the Plugfest. These recommendations are designed to help vendors, standards producers, and the SBOM community improve how SBOMs are generated so their results can be more consistent.

6.1 Recommendations for SBOM Minimum Elements

In our review of the submitted SBOMs, we found that some minimum elements were sporadically populated. As a result, we formed the following recommendations:

- **SBOM Type.** Emphasize including this attribute to document the lifecycle phase for which this SBOM was generated (e.g. Source, Build). We recommend that this attribute be required rather than optional because it is important for deciding which use cases the SBOM can support. Any tool should be able to report this attribute based on how it works and is invoked.
- **Component Version String.** Emphasize that accuracy in reporting exactly what the supplier provides is critical. Accurate reporting helps reduce the need for normalization when data is inconsistently reported (e.g., one SBOM reports *v 2.0* and another reports *2.0*). We also recommend that versions follow semantic versioning formats that allow some flexibility in reporting ranges of versions where necessary.
- **Component Supplier Name.** Emphasize the need for including the name of the entity that provided the contents of the software being described. This name helps users of the SBOM understand which third parties were part of the supply chain. For open source software components, which do not have a traditional supplier, a direct reference or link to the project repository should be provided.
- **Component Cryptographic Hash.** SBOM guidance should be clear about what is being hashed when a cryptographic hash is included. This guidance would make it more straightforward for SBOM users to know how to verify the hash value. Alternatively, SBOM creators should be explicit about what was hashed when supplying cryptographic hashes. For example, the hash may have been computed over a source file, a binary file, a compressed archive, etc.
- **Component License.** Emphasize the need to provide licensing information or to note that the license information is not known or was not included. Many submitted SBOMs did not include this field at all, which makes it difficult to know why it was not included (i.e. it might not be known or might have been considered out of scope).

6.2 Recommendations for SBOM Harmonization

We recommend the following to better harmonize SBOMs overall:

- **Normalization**. Develop recommendations about normalizing elements and their formats and forward them to the CycloneDX and SPDX standards teams.
- **Terminology**. Standardize on using the term *supplier* for a *primary supplier* and the term *manufacturer* for a *secondary supplier*.

- **Support Developer Community SBOM Efforts.** Some developer communities are working to include SBOM generators into language tools and build frameworks to make it much easier for projects using those languages and frameworks to generate SBOMs as upstream suppliers. These efforts have an outsize impact because they lower the barrier for creating SBOMs for every affected project. They also push the SBOM generation further upstream to project maintainers who have detailed knowledge of their own source code and build processes. The CPython community recently added SBOM support, for example [Python 2025]. Supporting these efforts has the potential to accelerate SBOM usage with a relatively low cost due to the economies of scale derived from SBOM tool use further upstream.
- **Dependencies**. Provide guidance to distinguish dependencies by category (e.g., runtime, tests, docs).
- **Component Inclusion Reason**. Standardize on annotating each component with the reason that it was included. This annotation would help users understand why a dependency (e.g., component, package, file) is included. Annotation could include an attribute or property (e.g., associated configuration and operating mode used to generate the SBOM) or different relationship types as applicable. Annotating in this way may not be possible for all dependencies, but it could be done when a tool has the context to provide that information. The following is an incomplete list of the reasons for including a component:
  - The component was included in a build manifest (e.g., pom.xml).
  - The component was used by a package manager during build.
  - The component was used by a platform toolchain during build.
  - The file was processed by a compiler.
  - The file was read to determine the license for a component.
- **SBOM Tools**. SBOM tools typically focus on a subset of the programming languages and build environments in use today. SBOM creators and users should be encouraged to ensure they are using an appropriate SBOM tool for their specific environment.
- **SBOM Profiles**. Interested stakeholders could develop and validate SBOM profiles[14] to ensure that each profile is useful and effective. Stakeholders can use SBOMs for a variety of different use cases (e.g., vulnerability identification, license compliance) and by a number of different communities (e.g., Health-ISAC and Auto-ISAC, health and automotive information-sharing analysis centers [Health-ISAC 2025, Auto-ISAC 202]).[15] Each combination of use case and community can be considered a context where stakeholders can use SBOMs to communicate among themselves. Stakeholders can use both the CycloneDX and SPDX standards for any given context because they are flexible enough to accommodate the required data with existing features. Improving interoperability within a given context requires restricting this flexibility so that SBOM artifacts can be shared and understood by all stakeholders. The OWASP Software Component Verifications Standard (SCVS) BOM Maturity

[14] An SBOM profile is a well-defined restriction placed on one or more SBOM standards to clarify the meaning and the allowable values for each field, its cardinality, and its other structural aspects.

[15] ISACs are Information Sharing and Analysis Centers. These centers are centralized sources of information about cybersecurity and security threats in a particular sector.

Model profiles feature is an example of such an approach [OWASP 2025e]. A simpler, and perhaps more pragmatic, approach would be to define a JSON schema that extends the existing JSON schemas for CycloneDX and/or SPDX and adds the necessary clarifications and restrictions for a profile.

6.3 Recommendations for Future Research

We recommend that the following areas be further researched:

- **Vulnerability Analysis**. Since vulnerability management is one popular use case of SBOMs, future research could focus on how well SBOMs support the vulnerability management function. This research could include how SBOMs support analyzing component vulnerabilities and how SBOMs support aligns with the National Vulnerability Database (NVD). Some Plugfest participants provided Vulnerability Exploitability eXchange (VEX) files for analysis, but these files were considered out of scope for the current Plugfest. Part of the challenge in vulnerability analysis is determining whether a vulnerable function within a component is being used since it is possible that vulnerabilities exist in the code but are not exploitable or consequential. This determination is critical to assessing true vulnerability and risk.
- **Analysis of SBOM Structure with Respect to Dependencies**. Analyzing the presence or absence of dependencies is only one dimension of understanding the structure of SBOMs.[16] The structure of dependency trees and alignment/variance in these structures can also provide insight into how the upstream effects of flaws in software components can percolate throughout dependent software packages and organizations. Researchers have begun conducting an introductory analysis of SBOMs to explore this facet of SBOM structure using dependency trees generated for assessing SBOM depth. We recommend conducting such an analysis to explore identifying discrepancies in SBOMs and their root causes. In Figure 4, we compared a Source CycloneDX SBOM and a Source SPDX SBOM for Hexyl from the same participant to check the following:[17]
  - Do the SBOMs contain the same components?
  - Are the overall structures of these two SBOMs and the relationships between the components identical?

  In this instance, we expected that the two SBOMs would have the same components and would contain the same relationships between components. Using an algorithm to match components and relationships across the two SBOMs resulted in a graphical representation of both SBOMs that confirmed our expectation. Applying this technique to other pairs or groups of SBOMs would highlight similarities and differences across SBOM structures.

[16] Analyzing dependencies can also help you understand the structure of software supply chains.

[17] This SBOM structure analysis was facilitated by code written by our colleagues at the Pacific Northwest National Laboratory [PNNL 2025].

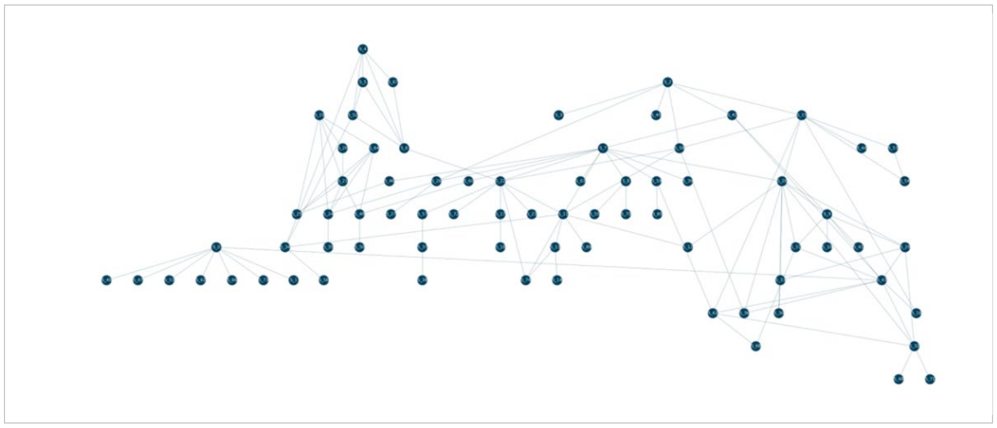

Figure 4: Dependency Tree Generated from Two Hexyl SBOMs

- **False Positives and False Negatives**. We need more research into what causes dependencies to be reported that are not actually included or used. If a component is reported as a dependency in an SBOM, then you should be confident that the component will be in your environment if you deploy that software. Likewise, if a component is not reported, you should be confident that the component will be absent from the environment if you deploy that software. We recommend conducting root cause analysis across multiple SBOMs to identify why components are or are not listed. This analysis would provide insight into when false positives and false negatives are detected in deployed software.
- **Dynamic Analysis**. We recommend conducting further research into how dynamic analysis at runtime used to resolve required native dependencies affects the number of dependencies in an SBOM.
- **SBOM Evolution**. SBOMs used in this Plugfest were static artifacts generated at a point in time. We recommend conducting additional research to better understand how to track and implement changes to SBOMs over time.

6.4 Recommendations for Improving Future Plugfests

We recommend the following improvements for future plugfests:

- **Security and Anonymity**. Allow anonymous submissions to the plugfest and provide stronger security for SBOM submission folders. We allowed individual contributors to configure their own security controls. Anyone who had concerns about security could email their submissions to us directly and then request us to delete those submissions when we no longer needed them.
- **Facilitate Sharing**. Some Plugfest participants were interested in seeing the SBOMs that others submitted to learn from them. Likewise, some participants may be interested in sharing their SBOMs with others. When there is interest, provide such access. We asked each participant for permission to make their SBOM submissions public. We are sharing the SBOMs from those who gave us their permission on GitHub at https://github.com/cmu-sei/sbom-plugfest-2024.

- **One-to-One Meetings with Plugfest Participants**. Plan to conduct some one-to-one meetings with individual participants as part of the plugfest process to better understand their SBOM generation processes and answer any questions they or the analysts may have. We conducted a few such meetings as time allowed.
- **Email List**. Establish an email list for the plugfest and allow participants to use it.
- **Use Case Plugfests**. For future plugfests, organizers should consider prescribing a specific use case or set of goals for SBOM submissions. Inspecting the harmonization (or lack thereof) of SBOMs submitted for the same use case may highlight different interpretations of requirements or methods for augmenting and enriching SBOM products.

Appendix: Detailed SBOM Reviews by Software Target

This Appendix details our analyses of the JSON SBOMs submitted for the nine individual software targets. For each target, our analyses include the following:

- a brief description of the software target function
- a chart summary of the numbers of SBOMs submitted
- a review of the required dependencies based on an inspection of the software as well as statistics on the percentages of SBOMs that included the declared dependencies
- summary “commonality” charts depicting the number of SBOMs that contained a given component of all components identified in the set of relevant SBOMs. These charts provide a visual sense of whether the SBOMs generally included the same components or diverged in their listed components. We generated these charts for Build and Source JSON SBOMs in both CycloneDX and SPDX format. Most would expect SBOMs of the same type for a given software target to capture the same dependencies. However, the charts depict variances in these SBOMs; most have a few common dependencies, but many dependencies are listed in only one or two SBOMs.
- a “commonality” chart covering the baseline components that we created using Syft, Grype, and the Microsoft SBOM tool. (We included this chart essentially for comparison purposes only to provide a sense of what we should expect the submitted SBOMs to include.)

HTTPie

HTTPie is a command-line HTTP client in Python designed for testing, debugging, and generally interacting with APIs and HTTP servers [HTTPie 2025].

Plugfest participants submitted 35 SBOMs in both Build and Source types, using both CycloneDX and SPDX standards, in both JSON and XML formats. None of the participants submitted SBOMs in YML format. Of the 35 SBOMs submitted, 30 were in JSON format.

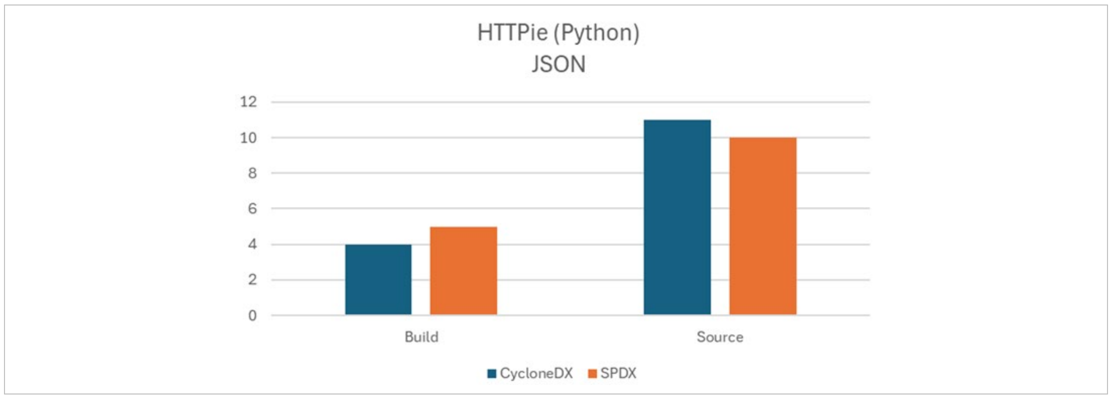

Figure 5: HTTPie Submissions by Type

Dependencies by Inspection

Our review of the code showed the following dependencies for HTTPie that were declared in the target repository's Python setup.cfg file. Table 9 shows the percentage of the submitted JSON SBOMs that contained the given declared dependency. Note that colorama is declared as a dependency only for the Windows 32-bit platform.

Table 9: Distribution of HTTPie SBOMs

Dependency	Version	CycloneDX		SPDX	
		Build %	**Source %**	**Build %**	**Source %**
pip	–	100	45	60	22
charset_normalizer	>=2.0.0	100	72	60	22
defusedxml	>=0.6.0	100	54	60	22
requests	>=2.22.0, <=2.31.0	100	63	60	22
Pygments	>=2.5.2	100	72	60	22
requests-toolbelt	>=0.9.1	100	9	60	22
multidict	>=4.7.0	100	63	60	22
setuptools	—	100	63	60	22
importlib-metadata	>=1.4.0	75	63	60	22
rich	>=9.10.0	100	63	60	22
colorama; sys_platform=="win32"	>=0.2.4	0	27	0	11

Component Commonality

We reviewed four Build and 11 Source CycloneDX SBOMs. We also reviewed five Build and 10 Source SPDX SBOMs. Figure 6 displays charts that show the number of SBOMs that contained a given component of all components identified in the set of SBOMs. We generated these charts for Build and Source SBOMs in both CycloneDX and SPDX format.

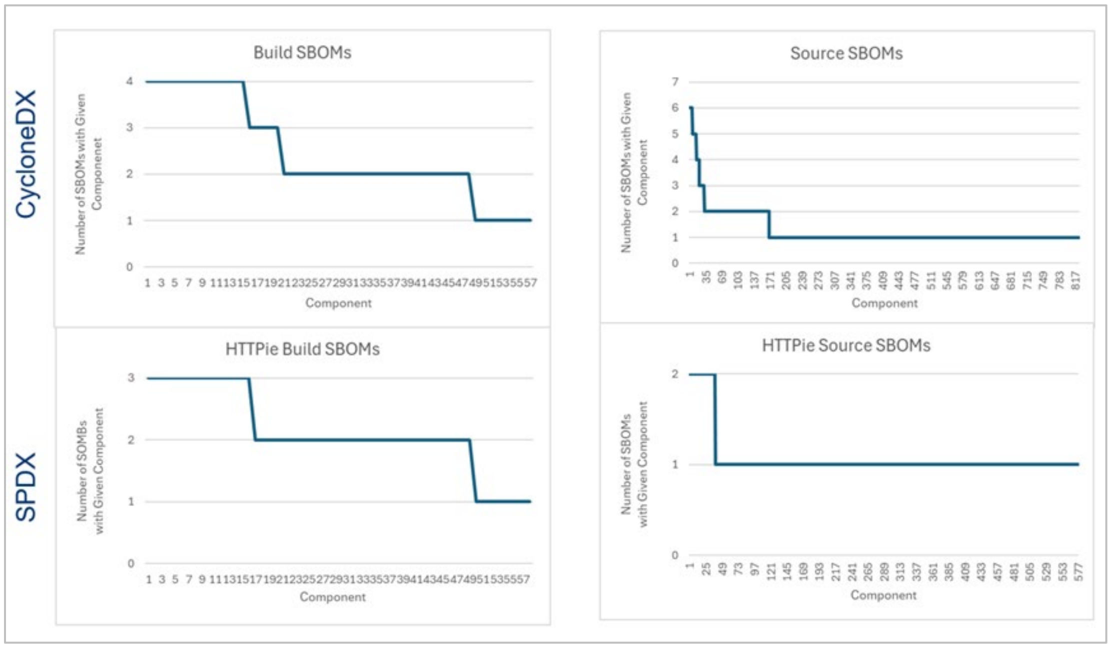

Figure 6: HTTPie Component Commonality

Baseline Commonality

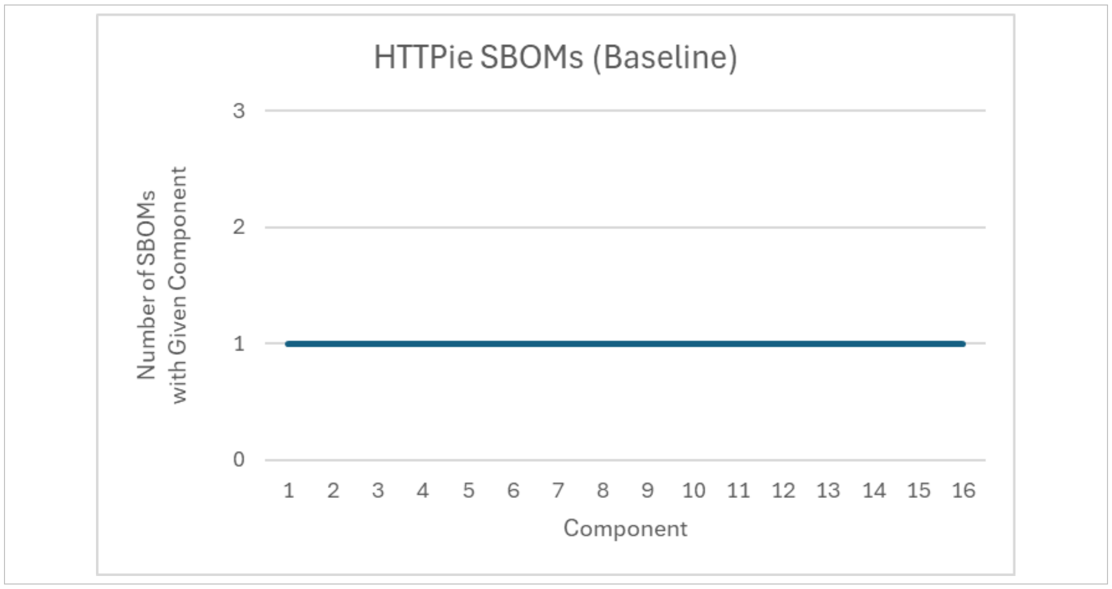

Figure 7: HTTPie Baseline SBOM Component Commonality

Only the Microsoft SBOM tool captured the dependencies for this target because it supported the setup.py file, and the other tools did not.

NodeJS-goof

NodeJS-goof is “a vulnerable Node.js demo application based on the Dreamers Lab tutorial” written in JavaScript [Snyk 2025].

Plugfest participants submitted 28 SBOMs in Build and Source types using both CycloneDX and SPDX standards and appearing in all three formats (JSON, SML, YML). Of the 28 SBOMs submitted, 20 were in JSON format.

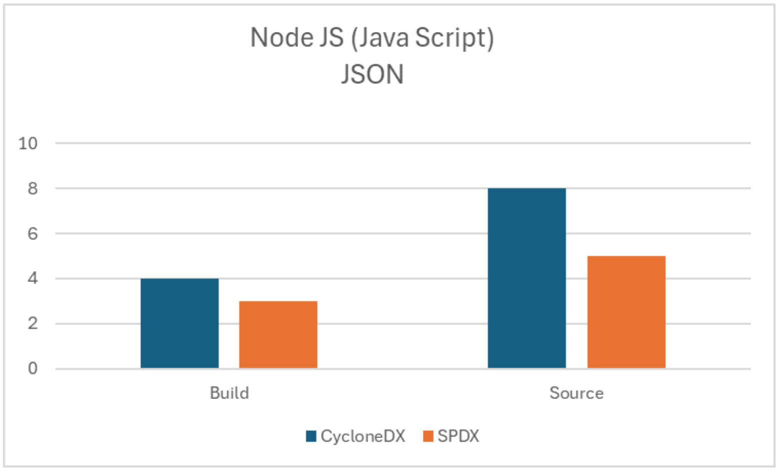

Figure 8: NodeJS Submissions by Type

Dependencies by Inspection

A review of the code showed the following declared dependencies in the package.json file for NodeJS-goof. Table 10 shows the percentage of the submitted JSON SBOMs that contained the given declared dependency.

Table 10: NodeJS-goof Dependencies by Inspection

Dependency	**Version**	**CycloneDX**		**SPDX**	
		Build %	**Source %**	**Build %**	**Source %**
adm-zip	0.4.7	100	87.5	67	100
body-parser	1.9.0	100	87.5	67	100
cfenv	^1.0.4	100	87.5	67	100
consolidate	0.14.5	100	87.5	67	100
dustjs-helpers	1.5.0	100	87.5	67	100
dustjs-linkedin	2.5.0	100	87.5	67	100
ejs	1.0.0	100	87.5	67	100
ejs-locals	1.0.2	100	87.5	67	100
errorhandler	1.2.0	100	87.5	67	100
express	4.12.4	100	87.5	67	100

Dependency	Version	CycloneDX		SPDX	
		Build %	Source %	Build %	Source %
express-fileupload	0.0.5	100	87.5	67	100
express-session	^1.17.2	100	87.5	67	100
file-type	^8.1.0	100	87.5	67	100
hbs	^4.0.4	100	87.5	67	100
humanize-ms	1.0.1	100	87.5	67	100
jquery	^2.2.4	100	87.5	67	100
lodash	4.17.4	100	87.5	67	100
marked	0.3.5	100	87.5	67	100
method-override	latest	100	87.5	67	100
moment	2.15.1	100	87.5	67	100
mongodb	^3.5.9	100	87.5	67	100
mongoose	4.2.4	100	87.5	67	100
morgan	latest	100	87.5	67	100
ms	^0.7.1	100	87.5	67	100
mysql	^2.18.1	100	87.5	67	100
npmconf	^2.18.1	100	87.5	67	100
typeorm	^0.2.24	100	87.5	67	100
validator	^13.5.2	100	87.5	67	100

Component Commonality

We reviewed four Build and eight Source CycloneDX SBOMs. We also reviewed three Build and five Source SPDX SBOMs. Figure 9 displays charts that show the number of SBOMs that contained a given component of all components identified in the set of SBOMs. We generated these charts for Build and Source SBOMs in both CycloneDX and SPDX format.

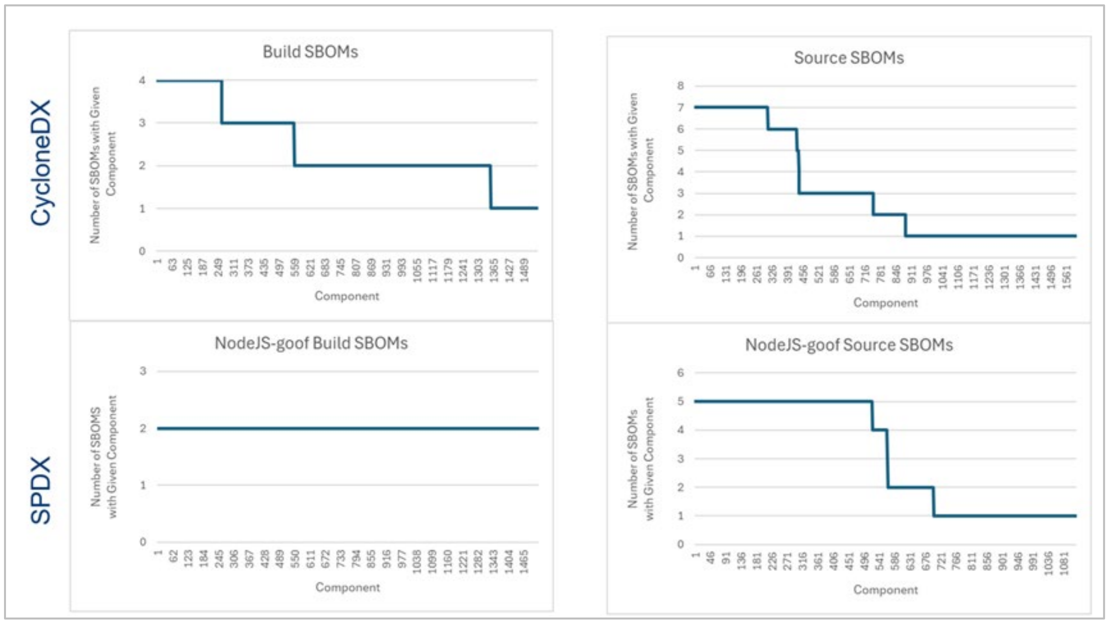

Figure 9: NodeJS-goof Component Commonality

Baseline Commonality

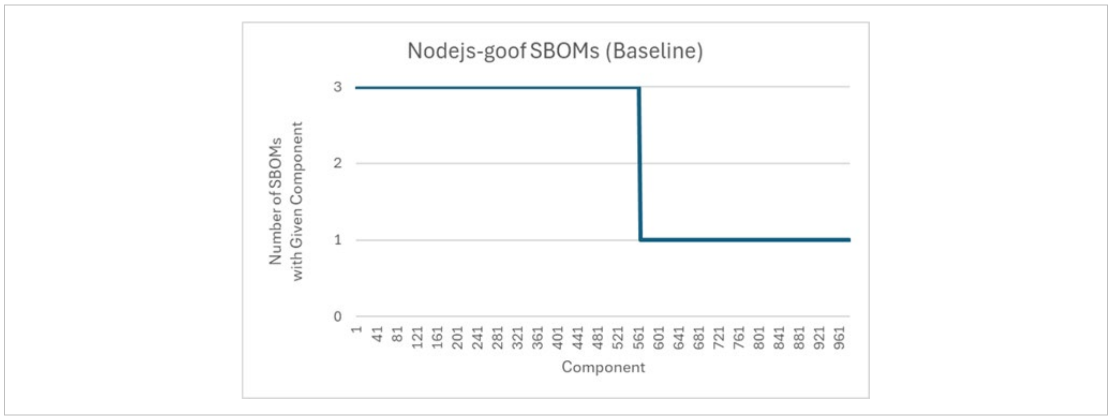

Figure 10: NodeJS-goof Baseline SBOM Component Commonality

Figure 10 illustrates the great commonality among the SBOMs generated by our baseline tools. All three SBOMs reported the same components for 564 components, and an additional 417 components were reported by just one SBOM.

MineColonies

The readme entry for MineColonies describes it as “an interactive building mod that allows you to create a thriving town within Minecraft” [IDTteam 2025]. MineColonies is written in Java.

Participants submitted 24 SBOMs in Build and Source types using both CycloneDX and SPDX standards and appearing in all three formats (JSON, SML, YML). Of the 24 SBOMs submitted, 17 were in JSON format.

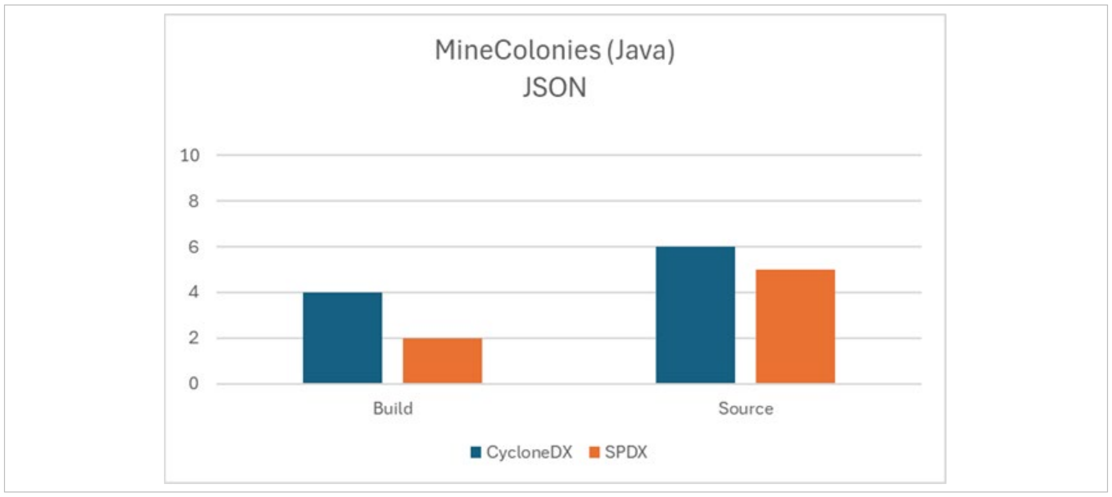

Figure 11: MineColonies Submissions by Type

Dependencies by Inspection

A review of the code showed the following declared dependencies in the dependencies.gradle file for MineColonies. Table 11 displays the percentage of the submitted JSON SBOMs that contained the given declared dependency.

This target follows the Gradle[18] conventions for declaring and building dependencies that are specific to the Minecraft mod community. Because it is specific to this community, the target may not be as widely supported by SBOM authors as more conventional Gradle targets.

Table 11: MineColonies Dependencies by Inspection

Dependency	Version	CycloneDX		SPDX	
		Build %	Source %	Build %	Source %
com.ldtteam:domum_ornamentum	1.20.1-1.0.184-BETA	25	17	0	20
com.ldtteam:blockui	1.20.1-1.0.139-BETA	25	17	0	20
com.ldtteam:structurize	1.20.1-1.0.740-BETA	25	17	0	20
com.ldtteam:multipiston	1.20-1.2.30-ALPHA	25	17	0	20
com.ldtteam:datagenerators	1.19.3-0.1.54-ALPHA	25	17	0	20

[18] For more information about Gradle, see https://gradle.org/.

Dependency	Version	CycloneDX		SPDX	
		Build %	Source %	Build %	Source %
mezz.jei:jei	15.1.0.19	25	34	0	20

Component Commonality

We reviewed four Build and six Source CycloneDX SBOMs. We also reviewed two Build and five Source SPDX SBOMs. Figure 12 displays charts that show the number of SBOMs that contained a given component of all components identified in the set of SBOMs. We generated these charts for Build and Source SBOMs in both CycloneDX and SPDX format.

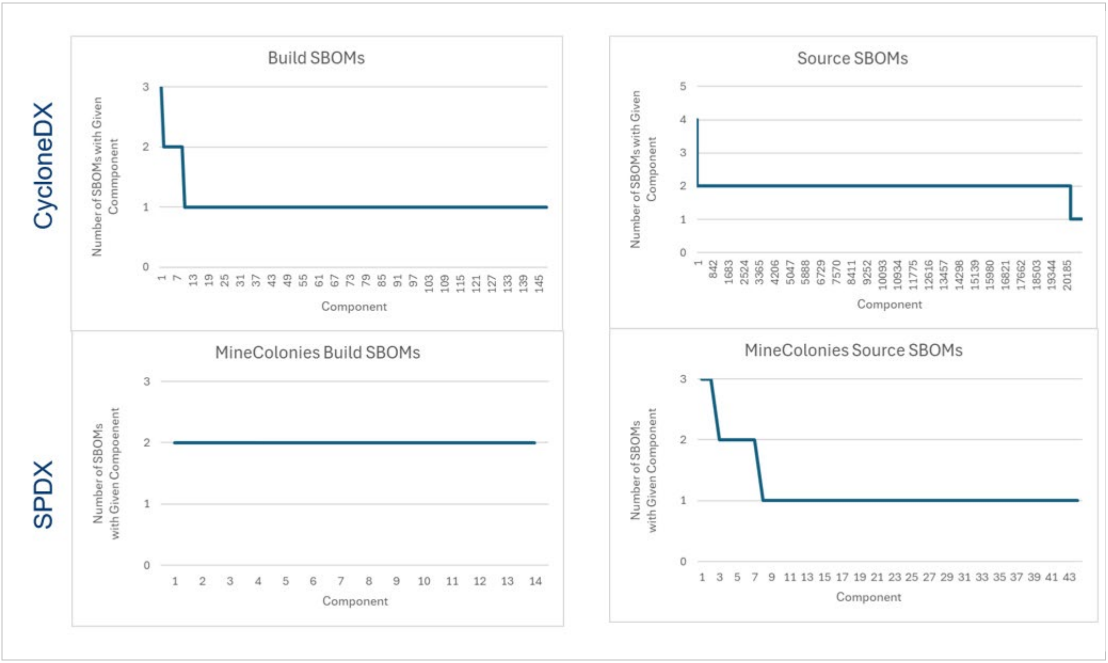

Figure 12: MineColonies Component Commonality

Baseline Commonality

We were not able to use any of our tools on this software target because it does not include the gradle.lockfile that Trivy and the Microsoft SBOM tool require for Gradle support, while Syft does not support Gradle at all.

Gin

Gin is a web framework written in Go (Golang) [Gin-Gonic 2025].

Plugfest participants submitted 23 SBOMs in Build and Source types using both CycloneDX and SPDX standards and appearing in all three formats (JSON, SML, YML). Of the 23 SBOMs submitted, 18 were in JSON format.

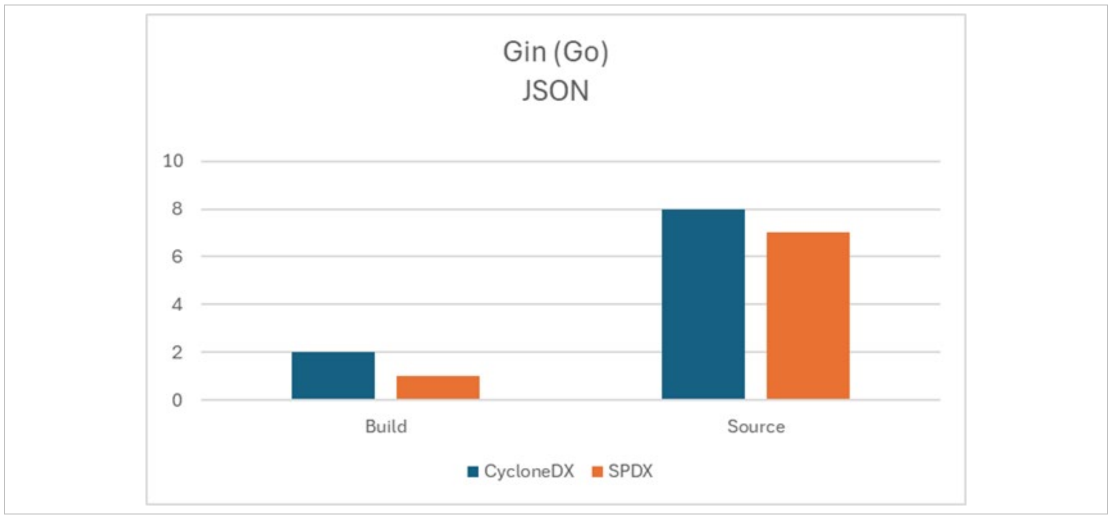

Figure 13: Gin Submissions by Type

Dependencies by Inspection

A review of the code showed the following declared dependencies in the go.mod file for Gin. Table 12 displays the percentage of the submitted JSON SBOMs that contained the given declared dependency.

Table 12: Gin Dependencies by Inspection

Dependency	Version	CycloneDX		SPDX	
		Build %	**Source %**	**Build %**	**Source %**
github.com/bytedance/sonic	v1.11.6	50	75	n/a	72
github.com/gin-contrib/sse	v0.1.0	100	75	n/a	72
github.com/go-playground/validator	v10.20.0	100	50	n/a	72
github.com/goccy/go-json	v0.10.2	50	75	n/a	72
github.com/json-iterator/go	v1.1.12	50	75	n/a	72
github.com/mattn/go-isatty	v0.0.20	100	75	n/a	72
github.com/pelletier/go-toml	v2.2.2	100	50	n/a	72
github.com/quic-go/quic-go	v0.43.1	100	75	n/a	72

Component Commonality

We reviewed two Build and eight Source CycloneDX SBOMs. We also reviewed seven Source SPDX SBOMs. Although one SPDX Build SBOM was submitted, its contents did not provide anything useful, so we were unable to use it to generate a chart. Figure 14 displays charts that show the number of SBOMs that contained a given component of all components identified in the set of

SBOMs. We generated these charts for Build and Source SBOMs in both CycloneDX and SPDX format.

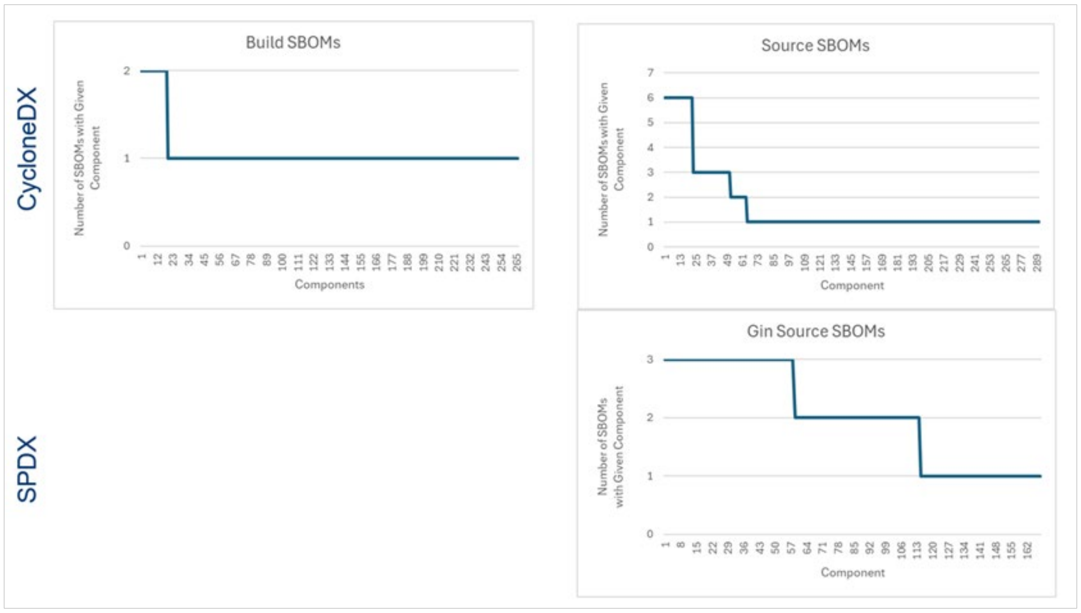

Figure 14: Gin Component Commonality

Baseline Commonality

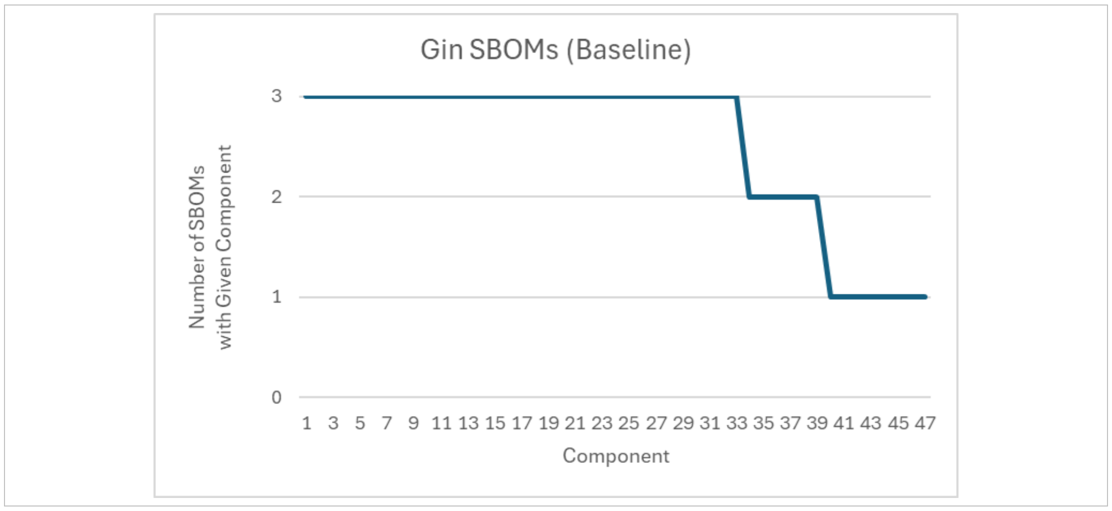

Figure 15: Gin Baseline SBOM Component Commonality

Figure 15 indicates that there was great commonality among the SBOMs generated by our baseline tools. All three SBOMs reported the same components except for a couple of components reported by just one SBOM.

Dependency Track

The readme entry for Dependency Track describes it as “an intelligent Component Analysis platform that allows organizations to identify and reduce risk in the software supply chain” [OWASP 2025d]. Dependency Track is written in Java.

Plugfest participants submitted 30 SBOMs in Build and Source types using both CycloneDX and SPDX standards and appearing in all three formats (JSON, SML, YML). Of the 30 SBOMs submitted, 22 were in JSON format.

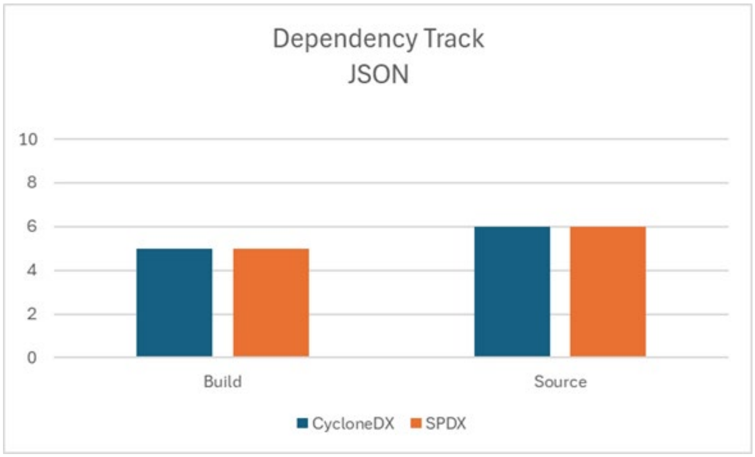

Figure 16: Dependency Track Submissions by Type

Dependencies by Inspection

A review of the code showed the following declared dependencies in the pom.xml file for Dependency Track. Table 13 displays the percentage of the submitted JSON SBOMs that contained the given declared dependency.

Table 13: Dependency Track Dependencies by Inspection

Dependency	**Version**	**CycloneDX**		**SPDX**	
		Build %	**Source %**	**Build %**	**Source %**
org.json/json	20240303	40	100	60	83
com.github.package-url/packageurl-java	1.5.0	40	100	60	33
org.apache.lucene/lucene-core	8.11.4	60	100	60	83
org.apache.lucene/lucene-analyzers-common	8.11.4	60	100	60	83
org.apache.lucene/lucene-queryparser	8.11.4	60	100	60	83
org.apache.lucene/lucene-queries	8.11.4	60	100	60	83

Dependency	Version	CycloneDX		SPDX	
		Build %	Source %	Build %	Source %
org.apache.lucene/lucene-sandbox	8.11.4	60	100	60	83
io.pebbletemplates/pebble	3.2.2	60	100	60	83
com.google.protobuf/proto-buf-java	4.28.2	60	100	60	83
com.google.protobuf/proto-buf-java-util	4.28.2	60	100	60	83
io.swagger.core.v3/swag-ger-jaxrs2-jakarta	2.1.22	40	100	60	66
org.apache.httpcompo-nents/httpclient	4.5.14	40	100	60	83
org.apache.httpcompo-nents.client5/httpclient5	5.4	40	100	60	83
org.apache.httpcompo-nents/httpmime	4.5.14	60	100	60	83
oauth.signpost/signpost-core	2.1.1	60	100	60	83
org.brotli/dec	0.1.2	40	100	60	83
com.fasterxml.wood-stox/woodstox-core	7.0.0	40	100	60	83
org.apache.maven/maven-artifact	3.9.9	60	100	60	83
com.mi-crosoft.sqlserver/mssql-jdbc	12.8.1.jre11	60	100	60	83
com.mysql/mysql-con-nector-j	8.2.0	40	100	60	83
org.postgresql/postgresql	42.7.4	40	100	60	83
com.google.cloud.sql/mysql-socket-factory-connector-j-8	1.20.1	40	100	60	83
com.google.cloud.sql/post-gres-socket-factory	1.20.1	40	100	60	83
com.google.cloud.sql/cloud-sql-connector-jdbc-sqlserver	1.20.1	40	100	60	83
org.apache.commons/com-mons-compress	1.27.1	60	100	60	83
org.apache.commons/com-mons-text	1.12.0	60	100	60	83
io.github.resilience4j/resili-ence4j-retry	2.2.0	40	100	60	83
io.github.resilience4j/resili-ence4j-ratelimiter	2.2.0	40	100	60	83
io.github.resilience4j/resili-ence4j-micrometer	2.2.0	40	100	60	83
org.slf4j/log4j-over-slf4j	2.0.16	60	100	60	83

<table>
<tr><th rowspan="2">Dependency</th><th rowspan="2">Version</th><th colspan="2">CycloneDX</th><th colspan="2">SPDX</th></tr>
<tr><th>Build %</th><th>Source %</th><th>Build %</th><th>Source %</th></tr>
<tr><td>org.kohsuke/github-api</td><td>1.323</td><td>40</td><td>100</td><td>60</td><td>83</td></tr>
</table>

Component Commonality

We reviewed five Build and six Source CycloneDX SBOMs. We also reviewed five Build and six Source SPDX SBOMs. Figure 17 displays charts that show the number of SBOMs that contained a given component of all components identified in the set of SBOMs. We generated these charts for Build and Source SBOMs in both CycloneDX and SPDX formats.

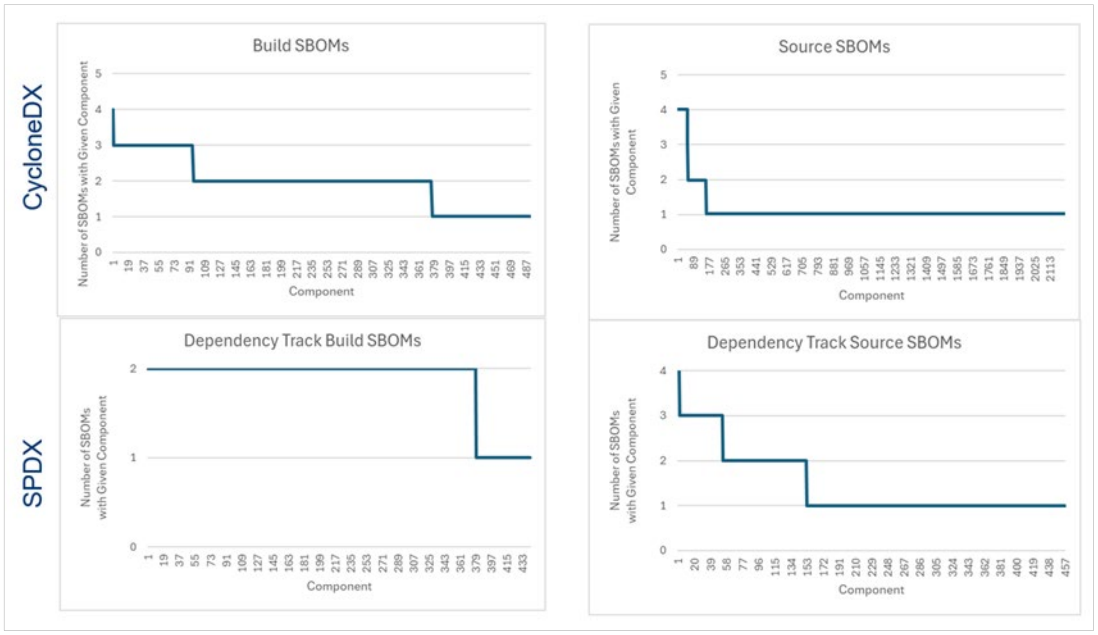

Figure 17: Dependency Track Component Commonality

Baseline Commonality

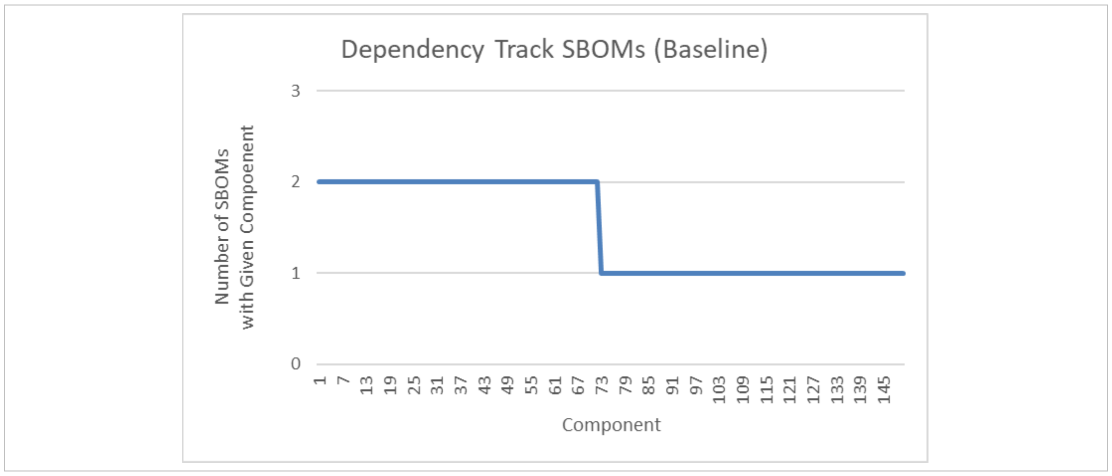

Figure 18: Dependency Track Baseline SBOM Component Commonality

The Syft and Trivy tools had more commonality than the chart displays because the names used in Trivy for this target included the group name as a prefix for the name, while Syft did not (e.g., commons-compress versus org.apache.commons:commons-compress). The Microsoft SBOM tool relies on the Maven[19] command-line tool (mvn) to parse dependencies in the Maven pom.xml. The mvn tool failed to process that file for this target.

PHPMailer

The readme entry for PHPMailer describes it as "a full-featured email creation and transfer class for PHP" [PHPMailer 2025]. PHPMailer is written in PHP, a general-purpose scripting language geared towards web development.

Plugfest participants submitted 21 SBOMs, all in Source type using both CycloneDX and SPDX standards and appearing in all three formats (JSON, SML, YML). Of the 21 SBOMs submitted, 16 were in JSON format.

19 For more information about Maven, see https://maven.apache.org/.

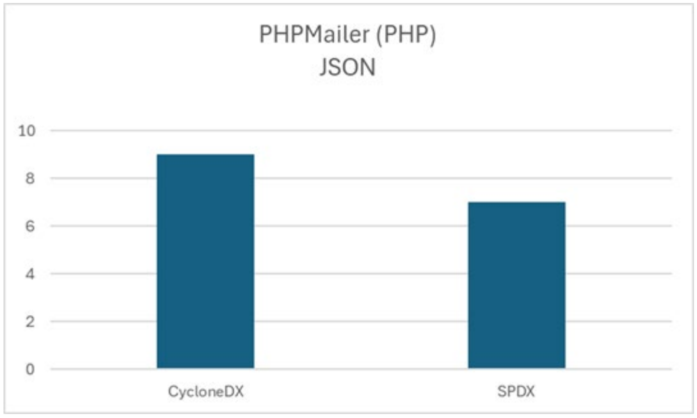

Figure 19: PHPMailer Submissions by Type

Dependencies by Inspection

A review of the code showed the following declared dependencies in the composer.json file for PHPMailer. Table 14 displays the percentage of the submitted JSON SBOMs that contained the given declared dependency.

Table 14: PHPMailer Dependencies by Inspection

Dependency	Version	CycloneDX		SPDX	
		Build %	**Source %**	**Build %**	**Source %**
ext-ctype	*	n/a	n/a	ext-ctype	*
ext-filter	*	n/a	n/a	ext-filter	*
ext-hash	*	n/a	n/a	ext-hash	*

This target includes only legacy dependencies that are now embedded in PHP, so no meaningful manual check is feasible.

Component Commonality

We reviewed nine Build and seven Source CycloneDX SBOMs. There were no SBOMs in SPDX format submitted. Figure 20 displays charts that show the number of SBOMs that contained a given component of all components identified in the set of SBOMs. We generated these charts for Source SBOMs in both CycloneDX and SPDX format. There were no Build SBOMs given the nature of PHPMailer.

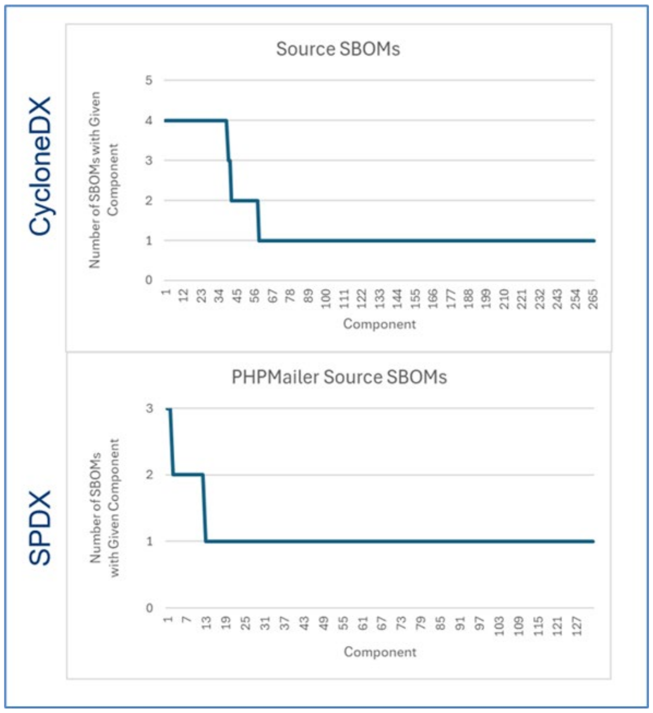

Figure 20: PHPMailer Component Commonality

Baseline Commonality

None of the baseline tools found dependencies because they either didn't support the PHP composer.json artifact or because the dependencies listed are all now part of the PHP language itself.

jq

The readme entry for jq describes it as "a lightweight and flexible command-line JSON processor akin to sed.awk,grep, and friends for JSON data. It's written in portable C and has zero runtime dependencies, allowing you to easily slice, filter, map, and transform structured data" [Jqlang 2025].

Plugfest participants submitted 32 SBOMs in Build and Source types using both CycloneDX and SPDX standards and appearing in all three formats (JSON, SML, YML). Of the 32 SBOMs submitted, 26 were in JSON format.

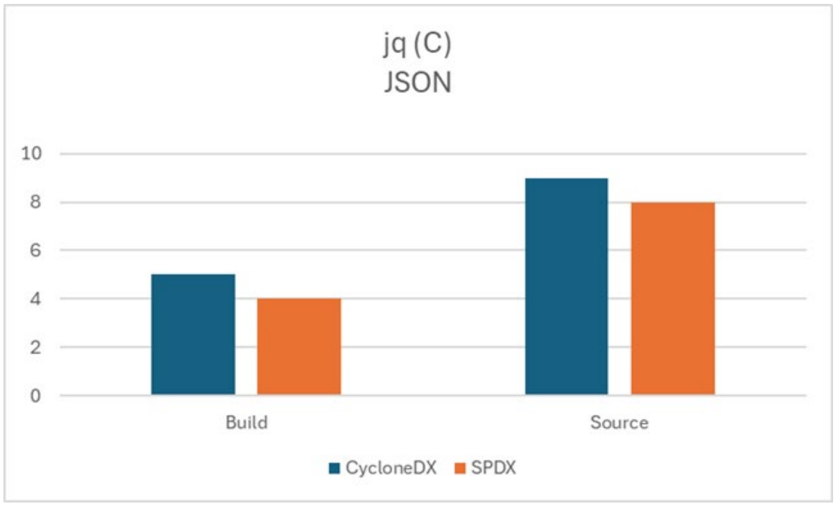

Figure 21: jq Submissions by Type

Dependencies by Inspection

A review of the code showed the following declared dependencies in the .gitmodules file for jq. Table 15 displays the percentage of the submitted JSON SBOMs that contained the given declared dependency.

Table 15: jq Dependencies by Inspection

Dependency	**Version**	**CycloneDX**		**SPDX**	
		Build %	**Source %**	**Build %**	**Source %**
oniguruma	nil	20	11	0	0

Note: This target has only one dependency, which is optional, so it is equally correct to include or exclude it.

Component Commonality

We reviewed five Build and nine Source CycloneDX SBOMs. We also reviewed four Build and eight Source SPDX SBOMs. Figure 22 displays charts that show the number of SBOMs that contained a given component of all components identified in the set of SBOMs. We generated these charts for Build and Source SBOMs in both CycloneDX and SPDX format.

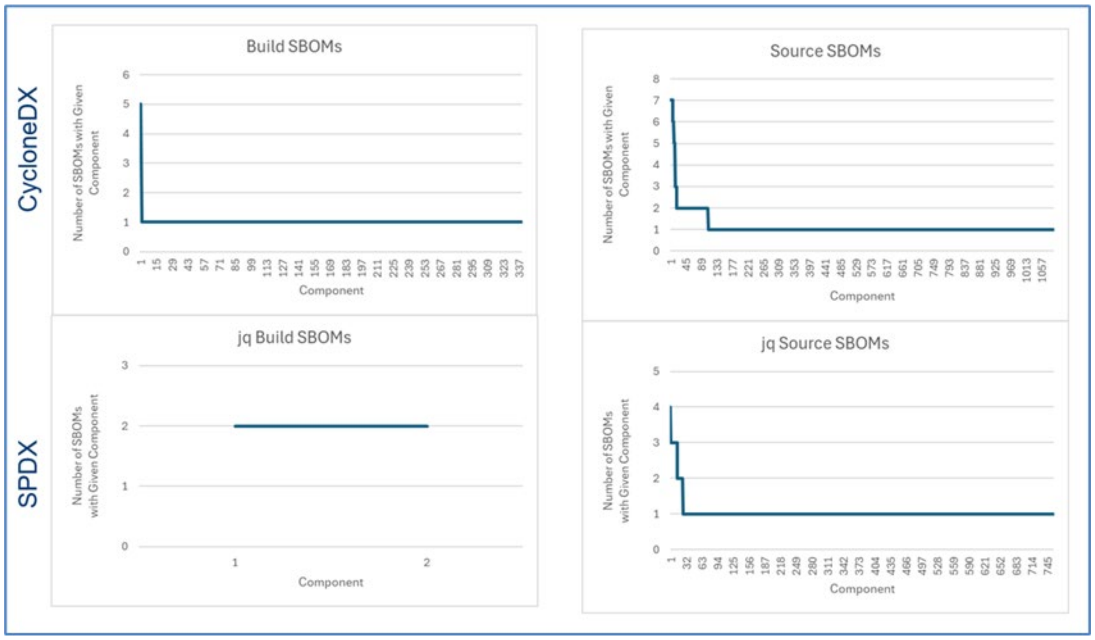

Figure 22: jq Component Commonality

Baseline Commonality

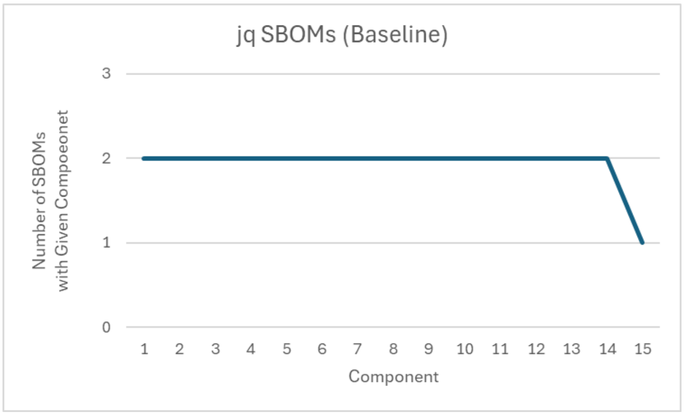

Figure 23: jq Baseline SBOM Component Commonality

The jq target does not have any explicit C-language dependencies, but it does include some Python dependencies that are used to build the project documentation. The Microsoft SBOM tool did not find these dependencies because it does not support the pip files that this target uses.

OpenCV

OpenCV.org describes OpenCV as “the world’s biggest computer vision library. OpenCV is open source, contains over 2500 algorithms, and is operated by the non-profit Open Source Vision Foundation” [OpenCV 2025].

Plugfest participants submitted 25 SBOMs in Build and Source types using both CycloneDX and SPDX standards and appearing in all three formats (JSON, SML, YML). Of the 25 SBOMs submitted, 20 were in JSON format.

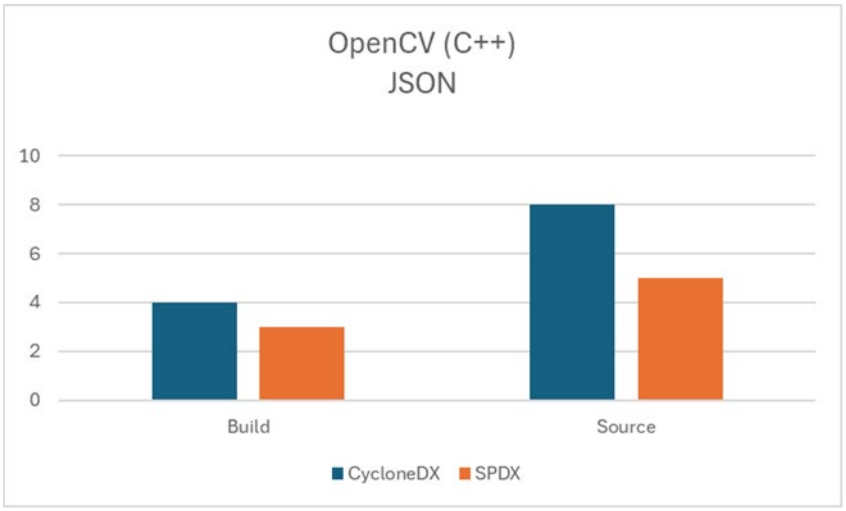

Figure 24: OpenCV Submissions by Type

Dependencies by Inspection

We did not manually inspect dependencies for OpenCV, because they are included using a complex make file with multiple options, and many dependencies are duplicated in the OpenCV repository itself.

Component Commonality

We reviewed four Build and eight Source CycloneDX SBOMs. We also reviewed three Build and five Source SPDX SBOMs. Figure 25 displays charts that show the number of SBOMs that contained a given component of all components identified in the set of SBOMs. We generated these charts for Build and Source SBOMs in both CycloneDX and SPDX format.

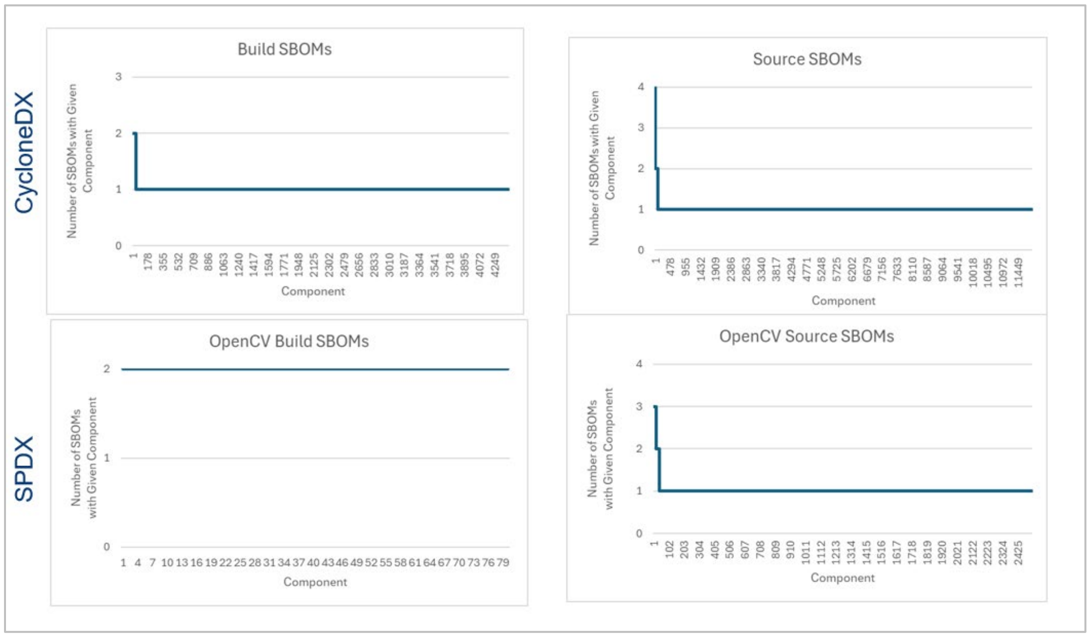

Figure 25: OpenCV Component Commonality

Baseline Commonality

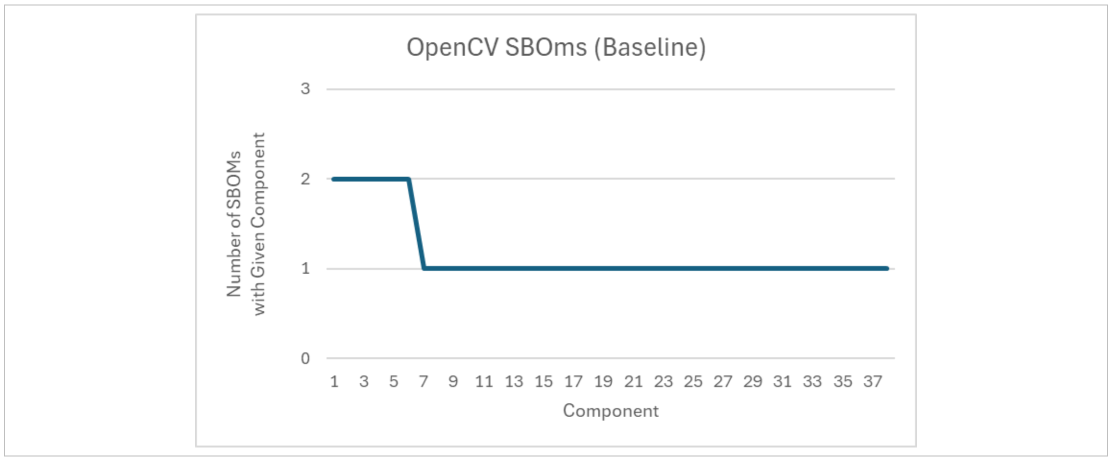

Figure 26: OpenCV Baseline SBOM Component Commonality

The baseline tools found various dependencies among the modules included within the source code repository. The presence or absence of these modules depends on what build options were chosen when building the library. None of these modules is a C++ library dependency except for the ones copied directly from their origin repository to the OpenCV repository.

Hexyl

The readme entry for Hexyl describes it as "a hex viewer for the terminal. It uses a colored output to distinguish different categories of bytes (NULL bytes, printable ASCII characters, ASCII whitespace characters, other ASCII characters and non-ASCII)" [Sharkdp 2025]. Hexyl is written in Rust.

Plugfest participants submitted 25 SBOMs in Build and Source types using both CycloneDX and SPDX standards and appearing in all three formats (JSON, SML, YML). Of the 25 SBOMs submitted, 17 were in JSON format.

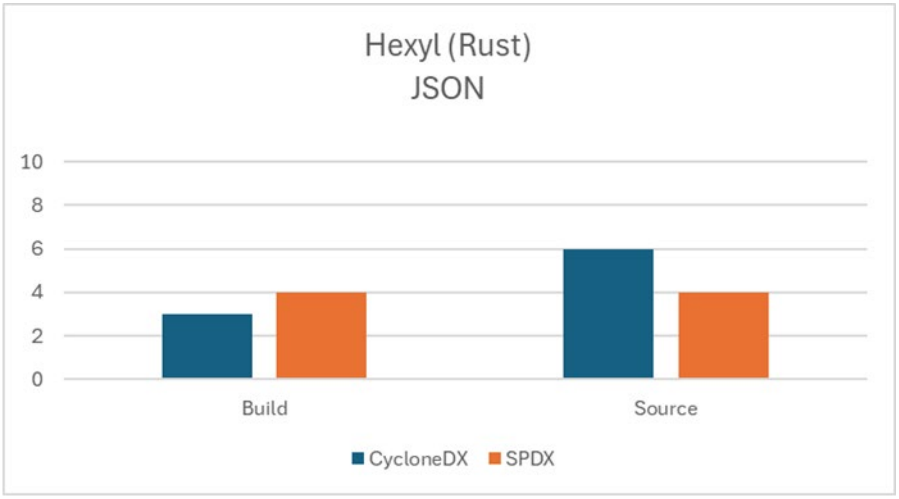

Figure 27: Hexyl Submissions by Type

Dependencies by Inspection

A review of the code showed the following declared dependencies in the cargo.toml file for Hexyl. Table 16 displays the percentage of the submitted JSON SBOMs that contained the given declared dependency.

Table 16: Hexyl Dependencies by Inspection

Dependency	**Version**	**CycloneDX**		**SPDX**	
		Build %	**Source %**	**Build %**	**Source %**
anyhow	1.0	100	100	75	100
const_format	0.2	100	100	75	100
libc	0.2	100	100	75	100
owo-colors	3	100	100	75	100
supports-color	2	100	100	75	100
thiserror	1.0	100	100	75	100
terminal_size	0.2	100	100	75	100

Component Commonality

We reviewed three Build and six Source CycloneDX SBOMs. We also reviewed four Build and four Source SPDX SBOMs. Figure 28 displays charts that show the number of SBOMs that contained a given component of all components identified in the set of SBOMs. We generated these charts for Build and Source SBOMs in both CycloneDX and SPDX format. We mapped both Build and Source SBOM components on the same chart with the first 78 components aligned. In this case, we were able to ensure the components were aligned across both SBOMs, since the number of components was relatively low.

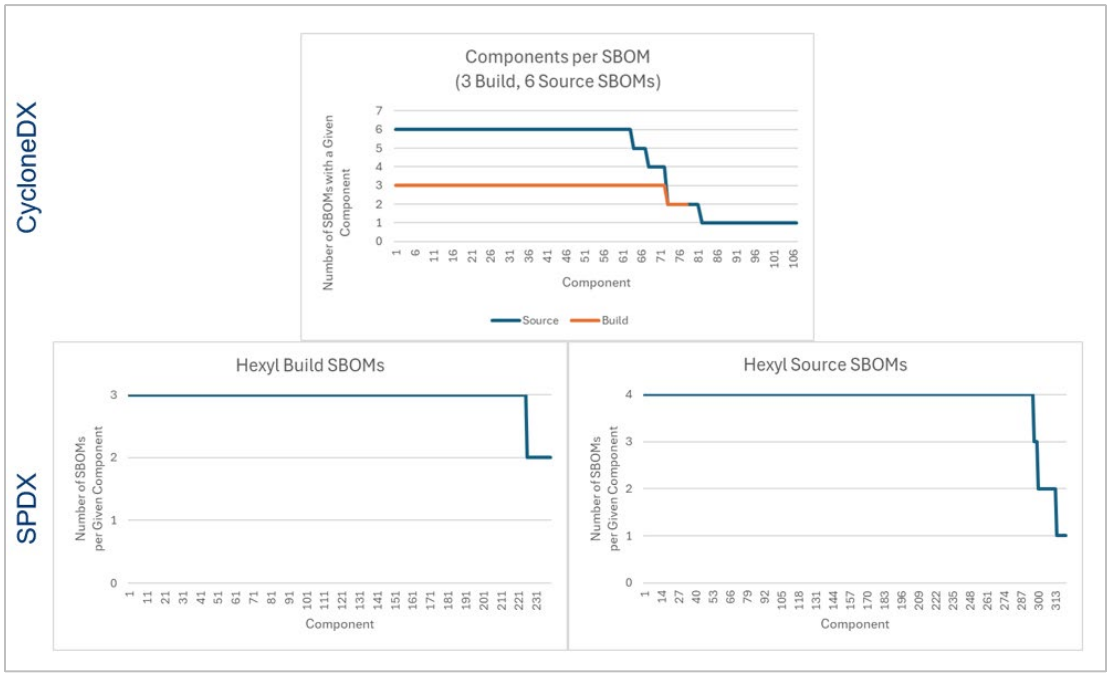

Figure 28: Hexyl Component Commonality

Baseline Commonality

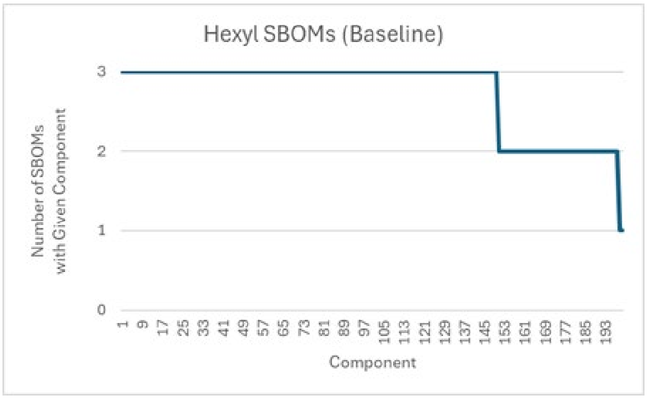

Figure 29: Hexyl Baseline SBOM Component Commonality

Figure 29 indicates that there was great commonality among the SBOMs generated by our baseline tools. All three SBOMs reported the same 149 components. We noticed what appears to be a bug in the Microsoft SBOM tool in reporting the purl of Rust cargo dependencies. All reported purl values included an extra forward slash and ended in the hash character (e.g., pkg:cargo//termtree@0.4.1# rather than pkg:cargo/termtree@0.4.1). Because of this bug, the Microsoft SBOM tool reported the last 48 counted components essentially in duplicate: once with the hash (#) and once without the hash. So, only 24 components actually existed. Participants reported these components this way for two SBOMs. (Figure 29 reflects this count in the right-hand quarter.)

Bibliography

URLs are valid as of the publication date of this report.

[Auto-ISAC 2025]
Automotive Information Sharing and Analysis Center (ISAC). *Auto-ISAC Website*. March 10, 2025 [accessed]. https://automotiveisac.com/

[Chase 2024]
Chase, Penny; Zuk, Margie; & Coley, Steven Christey. *Data Normalization Challenges and Mitigations in Software Bill of Materials (SBOM) Processing*. MITRE Corporation. October 24, 2024. https://www.mitre.org/news-insights/publication/data-normalization-challenges-mitigations-software-bill-materials-processing

[CIPAC 2023a]
Critical Infrastructure Partnership Advisory Council (CIPAC). *Securing the Software Supply Chain: Recommended Practices for Software Bill of Materials Consumption*. CIPAC. November 2023. https://www.cisa.gov/sites/default/files/2024-08/SECURING_THE_SOFTWARE_SUPPLY_CHAIN_RECOMMENDED_PRACTICES_FOR_SOFTWARE_BILL_OF_MATERIALS_CONSUMPTION-508.pdf

[CIPAC 2023b]
Critical Infrastructure Partnership Advisory Council (CIPAC). *Securing the Software Supply Chain: Recommended Practices for Managing Open-Source Software and Software Bill of Materials*. CIPAC. December 2023. https://media.defense.gov/2023/Dec/11/2003355557/-1/-1/0/ESF_SECURING_THE_SOFTWARE_SUPPLY_CHAIN%20RECOMMENDED%20PRACTICES%20FOR%20MANAGING%20OPEN%20SOURCE%20SOFTWARE%20AND%20SOFTWARE%20BILL%20OF%20MATERIALS.PDF

[CISA 2023]
Cybersecurity and Infrastructure Security Agency (CISA). *Types of Software Bill of Material (SBOM) Documents*. CISA. April 21, 2023. https://www.cisa.gov/sites/default/files/2023-04/sbom-types-document-508c.pdf

[CISA 2024]
Cybersecurity and Infrastructure Security Agency (CISA). *Framing Software Component Transparency: Establishing a Common Software Bill of Materials (SBOM), Third Edition*. CISA. September 3, 2024. https://www.cisa.gov/sites/default/files/2024-10/SBOM%20Framing%20Software%20Component%20Transparency%202024.pdf

[CISA 2025]
Cybersecurity and Infrastructure Security Agency (CISA). CISA Software Bill of Materials (SBOM). *CISA Website*. March 10, 2025 [accessed]. https://www.cisa.gov/sbom

[DOC 2021]
U.S. Department of Commerce (DOC). *The Minimum Elements for a Software Bill of Materials (SBOM)*. National Telecommunications and Information Administration (NTIA). July 12, 2021. https://www.ntia.doc.gov/files/ntia/publications/sbom_minimum_elements_report.pdf

[Gin-Gonic 2025]
Gin-Gonic. gin. *GitHub Website*. March 10, 2025 [accessed]. https://github.com/gin-gonic/gin

[Health-ISAC 2025]
Health Information Sharing and Analysis Center (ISAC). *Health-ISAC Website*. March 10, 2025 [accessed]. https://health-isac.org/

[HTTPie 2025]
HTTPie. HTTPie CLI. *GitHub Website*. March 10, 2025 [accessed]. https://github.com/HTTPie/cli

[IDTteam 2025]
IDTteam. minecolonies. *GitHub Website*. March 10, 2025 [accessed]. https://github.com/ldtteam/minecolonies

[ISO/IEC 2021]
International Organization for Standardization/International Electrotechnical Commission (ISO/IEC). *Information Technology—SPDX® Specification V2.2.1*. ISO/IEC 5962:2021. ISO/IEC. 2021. https://www.iso.org/standard/81870.html

[Jqlang 2025]
Jqlang. jq. *GitHub Website*. March 10, 2025 [accessed]. https://github.com/jqlang/jq

[Linux 2023]
The Linux Foundation. *Software Package Data Exchange (SPD™) Specification, Version 1.0*. The Linux Foundation. September 31, 2023. https://spdx.dev/wp-content/uploads/sites/31/2023/09/spdx-1.0.pdf

[Linux 2024]
The Linux Foundation. *Software Package Data Exchange (SPD™) Specification, Version 3.0.1*. The Linux Foundation. December 31, 2024. https://spdx.dev/wp-content/uploads/sites/31/2024/12/SPDX-3.0.1-1.pdf

[NTIA 2019]
National Telecommunications and Information Administration (NTIA). *Roles and Benefits for SBOM Across the Supply Chain*. NTIA. November 8, 2019. https://www.ntia.gov/files/ntia/publications/ntia_sbom_use_cases_roles_benefits-nov2019.pdf

[NTIA 2021a]
National Telecommunications and Information Administration (NTIA). *Survey of Existing SBOM Formats and Standards*. NTIA. 2021. https://www.ntia.gov/sites/default/files/publications/sbom_formats_survey-version-2021_0.pdf

[NTIA 2021b]
National Telecommunications and Information Administration (NTIA). *Software Identification Challenges and Guidance*. NTIA. March 30, 2021. https://www.ntia.gov/files/ntia/publications/ntia_sbom_software_identity-2021mar30.pdf

[NTIA 2025]
National Telecommunications and Information Administration (NTIA). SBOM Plugfest I Summary. Google Docs. April 21, 2025 [accessed]. https://docs.google.com/document/d/1vtqjjZIreY-lxFTTtwUSBD62CXEKu6T5pj7syxrJVcrA/edit?tab=t.0

[OASIS 2021]
Organization for the Advancement of Structured Information Standards (OASIS). Plugfest #2 – 20210622. *Google Drive*. June 22, 2021. https://drive.google.com/drive/u/0/folders/1Ujxp8w7dhrL6TNj5NxcaASbqPSoVTP1Y

[OpenCV 2025]
OpenCV Team. OpenCV. *OpenCV Website*. https://opencv.org/

[OWASP 2024]
Open Worldwide Application Security Project (OWASP) Foundation. CycloneDX v1.6: Now an Ecma International Standard. CycloneDX Website. July 1, 2024. https://cyclonedx.org/news/cyclonedx-v1.6-now-an-ecma-international-standard/

[OWASP 2025a]
The Open Worldwide Application Security Project (OWASP) Foundation. Dependency-Track Introduction. *Dependency Track Website*. March 10, 2025 [accessed]. https://docs.dependencytrack.org/

[OWASP 2025b]
The Open Worldwide Application Security Project (OWASP) Foundation. *CycloneDX*. OWASP. March 10, 2025 [accessed]. https://cyclonedx.org/guides/CycloneDX%20One%20Pager.pdf

[OWASP 2025c]
The Open Worldwide Application Security Project (OWASP) Foundation. CycloneDX: Dependency Relationship Compositions. *CycloneDX Website*. March 10, 2025 [accessed]. https://cyclonedx.org/use-cases/compositions-dependencies/

[OWASP 2025d]
The Open Worldwide Application Security Project (OWASP) Foundation. dependency-track. *GitHub Website*. March 10, 2025 [accessed]. https://github.com/DependencyTrack/dependency-track

[OWASP 2025e]
The Open Worldwide Application Security Project (OWASP) Foundation. BOM Maturity Model. *OWASP Website*. March 10, 2025 [accessed]. https://scvs.owasp.org/bom-maturity-model/

[PHPMailer 2025]
PHPMailer. PHPMailer. *GitHub Website*. March 10, 2025 [accessed]. https://github.com/PHP-Mailer/PHPMailer

[PNNL 2025]
Pacific Northwest National Laboratory (PNNL). graph_merge. *GitHub Website*. March 10, 2025 [accessed]. https://github.com/pnnl/graph_merge

[Python 2025]
Python Software Foundation. Software Bill-of-Materials Information. *Python Website*. March 10, 2025 [accessed]. https://www.python.org/downloads/metadata/sbom/

[Sharkdp 2025]
Sharkdp. hexyl. *GitHub Website*. March 10, 2025 [accessed]. https://github.com/sharkdp/hexyl

[Snyk 2025]
Snyk Limited. nodejs-goof. *GitHub Website*. March 10, 2025 [accessed]. https://github.com/snyk-labs/nodejs-goof

[White House 2021]
The White House. *Improving the Nation's Cybersecurity*. Executive Order 14028. U.S. National Archives. May 12, 2021. https://www.federalregister.gov/documents/2021/05/17/2021-10460/improving-the-nations-cybersecurity

REPORT DOCUMENTATION PAGE				*Form Approved OMB No. 0704-0188*
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.				
1. AGENCY USE ONLY (Leave Blank)	2. REPORT DATE May 2025			3. REPORT TYPE AND DATES COVERED Final
4. TITLE AND SUBTITLE Software Bill of Materials (SBOM) Harmonization Plugfest 2024				5. FUNDING NUMBERS FA8702-15-D-0002
6. AUTHOR(S) David Tobar, Jessie Jamieson, Mark Priest, Sasank Vishnubhatla, and Jason Fricke				
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Software Engineering Institute Carnegie Mellon University Pittsburgh, PA 15213				8. PERFORMING ORGANIZATION REPORT NUMBER CMU/SEI-2025-SR-002
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) SEI Administrative Agent AFLCMC/AZS 5 Eglin Street Hanscom AFB, MA 01731-2100				10. SPONSORING/MONITORING AGENCY REPORT NUMBER n/a
11. SUPPLEMENTARY NOTES				
12A DISTRIBUTION/AVAILABILITY STATEMENT Unclassified/Unlimited, DTIC, NTIS				12B DISTRIBUTION CODE
13. ABSTRACT (MAXIMUM 200 WORDS) This report describes the research findings and recommendations that resulted from the 2024 SBOM Harmonization Plugfest research project. The SEI project team managed the Plugfest and conducted research into the submitted software bills of material (SBOMs) in support of Cybersecurity & Infrastructure Security Agency (CISA). In this project, the SEI focused on understanding how differences in SBOM generation can result in different SBOM outputs. After gaining a better understanding of what causes these differences, the SEI project team developed recommendations for organizations to ensure more predictable and higher quality SBOMs. This report contains six major sections: an introduction, an explanation of the SBOM Plugfest process, an overview of SBOM submissions from participants, a description of the SEI project team's analysis, the team's findings, and the team's recommendations.				
14. SUBJECT TERMS SBOM				15. NUMBER OF PAGES 61
16. PRICE CODE				
17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified		20. LIMITATION OF ABSTRACT UL

NSN 7540-01-280-5500 Standard Form 298 (Rev. 2-89) Prescribed by ANSI Std. Z39-18 298-102