System & Software Security Software Bills of Materials Hands-On Project

Davide Baggio	s4426428
João Pereira	s4443322
Nuno Pereira	s4443330

November 2024

We confirm that this report was fully produced by the team members **Davide Baggio, João Pereira and Nuno Pereira** and we are jointly responsible for all content presented in this work. All used sources were attributed properly.

Contents

1 Introduction

Software Chain Security is an increasingly important challenge to tackle as early as possible in the Software Development process due to the catastrophic effect that vulnerabilities or security issues in downstream dependencies can cause to any software product, like in the SolarWinds [6] and Log4J [5] cases. It is therefore crucial from a development standpoint to be aware of what components are included in any software project and the potential vulnerabilities that they might introduce. Many solutions have been developed to aid in that process, the prime example being *package-managers* [1], tools that help managing software dependencies and version conflicts between dependencies. Examples of such package managers are NPM (Node Package Manager) and Cargo [21, 23].

Another solution to dependency and vulnerability tracking are *Software Bills of Materials* (SBOMs) [3], detailed listings of dependencies, their relations, licensing information and other metadata pertinent to software products. These provide a standard format to work and process dependency information, allowing easier communication and shareability. Currently, 3 mainstream SBOM standards exist: OWASP's *CycloneDX*, Linux Foundation's *SPDX* and NIST's *SWID* [12, 13, 19].

In previous work [25], the authors performed a critical comparison on 5 articles from the literature [7–11] based on author-defined metrics. The authors provide insights on the state-of-the-art regarding SBOMs and provide guidelines for SBOM adoption and development, as well as future work that can be done in further research.

In an attempt to complement [25], this article reports on a hands-on comparison of the three mainstream SBOM standards by making use of available tools for each standard. SBOM generation tools for each standard were used on a set of 3 major Open-Source repositories found on GitHub and the resulting SBOM output files were compared, both between standards as well as between the tools of each standard.

2 Dataset

In this section, we detail the dataset used to perform the hands-on comparison between SBOM standards. In 2.1 we discuss the different available standards and the tools chosen to generate SBOMs for each one of them are described in 2.2. Based on the tools picked we chose, in 2.3, the repositories for which the SBOMs will be generated.

2.1 Standards

Each one of the three standard formats focuses on a specific part of the software supply chain, which can be reflected in the (meta)data that each standard stores and processes. The tools developed for each each standard also reflect these decisions.

2.1.1 CycloneDX

CycloneDX [13] is an SBOM standard format developed by the CycloneDX Core Working Group and backed by the OWASP Foundation with a focus on "cyber-risk reduction" [13] and security [7]. The standard supports writing BOMs for several domains of software development, such as Software BOMs (SBOMs), Cryptographic BOMs (CBOMs), Software-as-a-Service BOMs (SaaSBOMs), among others. Over 200 tools related to CycloneDX's SBOM format are available at CycloneDX's official tool webpage.

For this hands-on comparison, we limited our search to *Open-Source* tools as these are free to access and use. Out of 172 listed Open-Source tools, 3 were chosen: CycloneDX `cdxgen` [14], `build-info-go` [18] and `syft` [17]. `syft` also supports SPDX, which will be discussed in 2.1.2.

Other tools exists but they are either unrelated (SBOM analysis, VEX generation, ...), too specific (official SBOM generators for several existing programming languages and build tools) or too limited on the supported development environments.

2.1.2 SPDX

SPDX (System Package Data Exchange) [19] is a standard format maintained by the Linux Foundation for communicating software bill of material information, including provenance, license, security, AI and other related information. All these characteristics make it a very versatile tool for software supply chain management. Nowadays it is used in the Linux kernel and in many package managers.

Tools used for SPDX SBOM generation include `syft` [17], ...

2.1.3 SWID

Software Identification (SWID) Tags [12] are a standard format for identifying software components and metadata, which can be used to generate SBOMs. Nowadays, the current standard of it is ISO/IEC 19770-2:2015 [2] and is maintained by the ISO/IEC JTC 1/SC 7/WG 21 committee [16].

A generated SWID Tag document consists of a well-organized collection of data fields that specify the software product, its version, identify the organizations and people involved in its creation and distribution, list the components that make up the software, define relationships between different software products and include additional metadata for further description. SWID tagging differs from CycloneDX and SPDX in that it is not a full-fledged SBOM format, as it doesn't aggregate information of all softwares so it's rather a standard for identifying software components and their metadata.

2.2 Tools

cdxgen is an official tool developed and released on GitHub by the CycloneDX team, built around the idea of being a "polyglot SBOM generator that is user friendly, precise and comprehensive". It provides a comprehensive SBOM generator for different versions of theCycloneDX Standard and also provides a Server mode, automatic licensing information and Docker/OCI container support. It is currently distributed as an NPM package.

build-info-go is a CLI tool to generate BuildInfo metadata, a custom format designed to encapsulate software components, their versions and their dependencies. The tool supports multiple languages and package managers and has the option to export the produced BuildInfo output into a valid CycloneDX JSON file.

syft is a "CLI tool and Go library for generating a Software Bill of Materials (SBOM) from container images and filesystems" [17]. It provides visibility into vulnerabilities and license compliance and can interact with modern vulnerability scanners such as Grype. It can output information in over 10 output formats, including custom user-defined formats specified by templates, and supports over 20 ecosystems.

swid-builder a Java API for building SWID and CoSWID tags. This library provides a set of builder patterns that can be used together to generate tags, however it is not a standalone tool and the provided documentation is not very clear on how to use it. Thus, it is not suitable for our use case.

swid-maven-plugin is a Maven plugin published by NIST [20] that generates SWID Tags for Java projects and is compliant with the above mentioned ISO standard. It needs to be configured in the project's `pom.xml` file and additionally it requires the assembly descriptor in `src/assembly/bin.xml` to be configured aswell. The plugin is not maintained anymore but still produces a valid SWID Tag file.

swid-generator as alternatives to the official tools, we also looked for unofficial ones on GitHub and found a tool developed by Labs64 [4], which also supports Gradle projects. The downside of it is that it requires implementing a custom portion of code in the project to generate the SWID Tags, which is not as straightforward as the other tool and it might be unsuitable for complex projects.
By the same authors, there is also a Maven plugin which is unfortunately not maintained anymore and uses an outdated version of Java, which results in a lot of errors when trying to build it, as testified by this issue.

2.3 Repositories

To ensure a fair comparison between standards, we chose a representative set of major Open-Source repositories that could be analyzed by most, if not all, of the tools selected.
As such, we have picked 3 repositories from GitHub:

- Apache Kafka [15] is a modern event-streaming platform written in Java by the Apache Software Foundation. Initially developed at Linkedin to accommodate their growing message processing needs, it now serves as the backbone for many asynchronous, event-driven and streaming systems around the world. This might change to another, Maven-based project
- numpy [22] is a Python library that provides a "multi-dimensional array object" and utilities and operations built around array objects for fast manipulation. Currently it supports most of the mainstream scientific and machine learning packages available, like SciPy and SciKit Learn.
- Kubernetes [24] is a container orchestration tool initially developed at Google to mimic their internal system Borg. It supports user defined-workloads and maintains a constant system state, ensuring high-availability and reliability.

3 Methodology

In order to effectively and correctly compare the three SBOM standards, we used the tools mentioned in Section 2.2 on each one of the three example projects. This process involved building the tools themselves and running them against each project according to project-specific configuration parameters. The resulting output files were grouped according to each standard, with the filename indicating the tool used for better analysis.

To aid this process, a custom *Bash* script was developed. This script is responsible for building each tool, applying the correct flags to the resulting executable and generating an SBOM for each example repository.

This process resulted in 12 reports for CycloneDX (it is worh noting that `syft` supports generating SBOMs from both files and directories, so these differences were taken into account), N files for SPDX and 0 files for SWID. A discussion of these results is present in Section 4.

4 Results and Discussion

4.1 CycloneDX

4.2 SPDX

4.3 SWID

Generating SWID tags turned out to be an unfeasible task due to the lack of proper tools and documentation. The tools provided by NIST only support Java projects built with Maven, which is too restrictive since the largest projects use Gradle nowadays, while the tools found on GitHub require implementing a custom portion of code in the project to generate the tags, which is not as straightforward as the other tools and it might be unsuitable for complex projects. In addition to this, the documentation provided by NIST is often not very clear on how to use the tools, which makes it even harder to generate the SWID tags.

Out of all the previosly mentioned tools, only `swid-maven-plugin` was able to generate a tag file for this Maven project. We also tried to use `swid-generator` to generate a SBOM for Kafka, since it is written in Java using Gradle. However, unfortunately this turned out to not be possible for the reasons mentioned above.

5 Conclusion

...

The standard that nowadays has the most critical aspects in usage is SWID, as testified by our unsuccessful attempts to generate SWID tags and the lack of variety in tools available for this purpose. This is a clear indicator that this standard is not as widely adopted as the others, as also shown in this previous research [11]. This is a problem that needs to be addressed, as SWID is a useful standard for software identification and its usage should be encouraged by making it easier to generate the tags.

References

[1] Diomidis Spinellis. "Package Management Systems". In: *IEEE Software* 29.2 (2012). [Accessed 21-10-2024], pp. 84–86. DOI: 10.1109/MS.2012.38.

[2] ISO Central Secretary. *Information technology — IT asset management – Part 2: Software identification tag*. en. Standard ISO/IEC 19770-2:2015. [Accessed 26-11-2024]. Geneva, CH: International Organization for Standardization, 2015. URL: https://www.iso.org/standard/65666.html.

[3] Éamonn Ó Muirí. "Framing software component transparency: Establishing a common software bill of material (SBOM)". In: *NTIA, Nov* 12 (2019). [Accessed 15-10-2024].

[4] Labs64. *Swid-generator*. https://github.com/Labs64/swid-generator. 2020.

[5] Ravie Lakshmanan. *Extremely Critical Log4J Vulnerability Leaves Much of the Internet at Risk*. [Accessed 27-11-2024]. 2021. URL: https://thehackernews.com/2021/12/extremely-critical-log4j-vulnerability.html.

[6] Sean Peisert et al. "Perspectives on the SolarWinds Incident". In: *IEEE Security & Privacy* 19.2 (2021), pp. 7–13. DOI: 10.1109/MSEC.2021.3051235.

[7] Boming Xia et al. "An Empirical Study on Software Bill of Materials: Where We Stand and the Road Ahead". In: *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*. [Accessed 15-10-2024]. 2023, pp. 2630–2642. DOI: 10.1109/ICSE48619.2023.00219.

[8] Nusrat Zahan et al. "Software Bills of Materials Are Required. Are We There Yet?" In: *IEEE Security & Privacy* 21.2 (2023). [Accessed 15-10-2024], pp. 82–88. DOI: 10.1109/MSEC.2023.3237100.

[9] Tingting Bi et al. "On the Way to SBOMs: Investigating Design Issues and Solutions in Practice". In: *ACM Trans. Softw. Eng. Methodol.* 33.6 (June 2024). [Accessed 15-10-2024]. ISSN: 1049-331X. DOI: 10.1145/3654442. URL: https://doi.org/10.1145/3654442.

[10] Berend Kloeg et al. "Charting the Path to SBOM Adoption: A Business Stakeholder-Centric Approach". In: *Proceedings of the 19th ACM Asia Conference on Computer and Communications Security*. ASIA CCS '24. [Accessed 15-10-2024]. Singapore, Singapore: Association for Computing Machinery, 2024, pp. 1770–1783. ISBN: 9798400704826. DOI: 10.1145/3634737.3637659. URL: https://doi.org/10.1145/3634737.3637659.

[11] Trevor Stalnaker et al. "BOMs Away! Inside the Minds of Stakeholders: A Comprehensive Study of Bills of Materials for Software Systems". In: *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*. ICSE '24. [Accessed 15-10-2024]. Lisbon, Portugal: Association for Computing Machinery, 2024. ISBN: 9798400702174. DOI: 10.1145/3597503.3623347. URL: https://doi.org/10.1145/3597503.3623347.

[12] National Institute of Standards and Technology (NIST). *Software Identification (SWID) Tagging — CSRC — CSRC — csrc.nist.gov*. https://csrc.nist.gov/projects/Software-Identification-SWID. [Accessed 15-10-2024].

[13] Open Worldwide Application Security Project (OWASP). *OWASP CycloneDX Software Bill of Materials (SBOM) Standard — cyclonedx.org*. https://cyclonedx.org/. [Accessed 15-10-2024].

[14] CycloneDX Core Team. *cdxgen documentation — cyclonedx.github.io*. [Accessed 24-11-2024]. URL: https://cyclonedx.github.io/cdxgen/.

[15] Apache Foundation. *GitHub - apache/kafka: Mirror of Apache Kafka — github.com*. https://github.com/apache/kafka. [Accessed 25-11-2024].

[16] Ms Reena Garg. *ISO/IEC JTC 1/SC 7*. https://www.iso.org/committee/45086.html. [Accessed 26-11-2024].

[17] *GitHub - anchore/syft: CLI tool and library for generating a Software Bill of Materials from container images and filesystems — github.com*. https://github.com/anchore/syft. [Accessed 29-11-2024].

[18] *JFrog Build Info - Build Info by JFrog — buildinfo.org*. [Accessed 24-11-2024]. URL: https://www.buildinfo.org/.

[19] Linux Foundation. *SPDX Linux Foundation Projects Site — spdx.dev*. https://spdx.dev/. [Accessed 15-10-2024].

[20] NIST. *swid-maven-plugin*. [Accessed 27-11-2024]. URL: https://pages.nist.gov/swid-tools/swid-maven-plugin/.

[21] NodeJS Team. *NPM: Node Package Manager*. [Accessed 15-10-2024]. URL: https://www.npmjs.com/.

[22] Numpy. *GitHub - numpy/numpy: The fundamental package for scientific computing with Python. — github.com*. https://github.com/numpy/numpy. [Accessed 28-11-2024].

[23] Rust Team. *Cargo*. [Accessed 15-10-2024]. URL: https://doc.rust-lang.org/cargo/.

[24] Kubernetes SIG. *GitHub - kubernetes/kubernetes: Production-Grade Container Scheduling and Management — github.com*. https://github.com/kubernetes/kubernetes. [Accessed 28-11-2024].

[25] Davide Baggio, João Pereira, and Nuno Pereira. “System & Software Security - Software Bills of Materials”. [Accessed 24-11-2024]. N.D.

A Appendix A

The code and data used to perform this work can be found at

https://github.com/naapperas/uleiden-sss