A Dataset of Software Bill of Materials for Evaluating SBOM Consumption Tools

Rio Kishimoto*, Tetsuya Kanda†, Yuki Manabe‡, Katsuro Inoue§, Shi Qiu¶, Yoshiki Higo*
*The University of Osaka, Japan, {r-kisimt, higo}@ist.osaka-u.ac.jp
†Notre Dame Seishin University, Japan, kanda@m.ndsu.ac.jp
‡The University of Fukuchiyama, Japan, manabe-yuki@fukuchiyama.ac.jp
§Nanzan University, Japan, inoue599@nanzan-u.ac.jp
¶Toshiba Corporation, Japan, shi1.qiu@toshiba.co.jp

***Abstract*—A Software Bill of Materials (SBOM) is becoming an essential tool for effective software dependency management. An SBOM is a list of components used in software, including details such as component names, versions, and licenses. Using SBOMs, developers can quickly identify software components and assess whether their software depends on vulnerable libraries. Numerous tools support software dependency management through SBOMs, which can be broadly categorized into two types: tools that generate SBOMs and tools that utilize SBOMs. A substantial collection of accurate SBOMs is required to evaluate tools that utilize SBOMs. However, there is no publicly available dataset specifically designed for this purpose, and research on SBOM consumption tools remains limited. In this paper, we present a dataset of SBOMs to address this gap. The dataset we constructed comprises 46 SBOMs generated from real-world Java projects, with plans to expand it to include a broader range of projects across various programming languages. Accurate and well-structured SBOMs enable researchers to evaluate the functionality of SBOM consumption tools and identify potential issues. We collected 3,271 Java projects from GitHub and generated SBOMs for 798 of them using Maven with an open-source SBOM generation tool. These SBOMs were refined through both automatic and manual corrections to ensure accuracy, currently resulting in 46 SBOMs that comply with the SPDX Lite profile, which defines minimal requirements tailored to practical workflows in industries. This process also revealed issues with the SBOM generation tools themselves. The dataset is publicly available on Zenodo (DOI: 10.5281/zenodo.14233414).**

***Index Terms*—SBOM, SPDX**

I. Introduction

Managing software dependencies is an increasingly important task due to its growing complexity. Inadequate software dependency management has resulted in delayed responses to vulnerabilities hidden within dependent libraries [1], [7], [17]. As a promising tool for addressing these challenges, a Software Bill of Materials (SBOM) has gained attention [10], [16]. An SBOM is a comprehensive inventory of components used in software, including details such as component names, versions, and licenses. Using SBOMs, developers can quickly identify the components their software depends on and assess whether these dependencies include vulnerable libraries.

Numerous tools currently support software dependency management through SBOMs, which can be broadly categorized into two types: tools that generate SBOMs and tools that consume (utilize) SBOMs. SBOM generation tools analyze software source code, extract information about its components, and produce an SBOM. SBOM consumption tools, on the other hand, analyze an SBOM to provide insights. For example, grype [2] is an SBOM consumption tool that accepts SBOMs as input and identifies vulnerabilities in the software components described within.

While the performance and limitations of SBOM generation tools have been studied extensively [3], [6], [14], [15], [18], research on SBOM consumption tools remains limited. One of the reasons is the lack of a publicly available dataset specifically designed for evaluating SBOM consumption tools. To effectively evaluate SBOM consumption tools, a substantial collection of accurate and well-structured SBOMs is necessary. The examples of SBOM [5], [12] typically describe software with few dependencies, limiting their utility for evaluating tools. Chainguard also provides a collection of SBOMs [4], but these are often automatically generated or obtained from the Internet, which means their accuracy cannot be guaranteed.

To address this gap, we present a dataset of SBOMs specifically designed to evaluate SBOM consumption tools, with a focus on two major SBOM applications: vulnerability management and license management. We collected 3,271 Java projects from GitHub and generated SBOMs for 798 of them using Maven with an open-source SBOM generation tool. These SBOMs were refined through both automatic and manual corrections to ensure accuracy, currently resulting in 46 SBOMs that comply with the SPDX Lite profile [13], which defines minimal requirements tailored to practical workflows in industries. These accurate and well-structured SBOMs enable researchers to thoroughly evaluate the functionality of SBOM consumption tools and identify potential issues.

II. Dataset Requirements and SBOM Format

There are two major SBOM formats: SPDX [11] and CycloneDX [9]. Code. 1 shows an excerpt of an SBOM written in the SPDX format. This example describes information about a widely used Java library, log4j-core version 2.10.0. For instance, the library name (name field, line 1), its version (versionInfo field, line 8), and its license (licenseConcluded

Code. 1
EXCERPT OF AN SBOM IN SPDX FORMAT FOR A JAVA PROJECT

```
 1 {
 2   "name": "log4j-core",
 3   "SPDXID": "SPDXRef-Package-log4net",
 4   "downloadLocation": "https://repo1.maven.org/maven2/o
         rg/apache/logging/log4j/log4j-core/2.10.0/log4j-
         core-2.10.0.jar",
 5   "filesAnalyzed": false,
 6   "licenseConcluded": "Apache-2.0",
 7   "copyrightText": "NOASSERTION",
 8   "versionInfo": "2.10.0",
 9   "externalRefs": [{
10     "referenceCategory": "PACKAGE-MANAGER",
11     "referenceLocator": "pkg:maven/org.apache.logging.lo
           g4j/log4j-core@2.10.0",
12     "referenceType": "purl"
13   }],
14   "supplier": "Organization: The Apache Software Founda
         tion",
15   "homepage": "https://logging.apache.org/ ... "
16 }
```

field, line 6) are recorded in the corresponding fields. SPDX defines numerous additional fields to represent various types of software-related information, and CycloneDX offers a similar set of fields. Since most of these fields are optional, the choice of which fields to include in an SBOM depends on its intended use case.

The primary use case of our dataset is the evaluation of SBOM consumption tools, with a focus on two major SBOM applications: vulnerability management and license management. Therefore, it is sufficient for the SBOMs in our dataset to include the essential information required for these two use cases.

In this paper, we adopt the SPDX format because it provides a set of mandatory fields called the SPDX Lite profile [13]. This profile defines a minimal set of fields tailored to actual workflows in industries. By complying with this profile, we ensure that the SBOMs in our dataset contain the critical information needed for the major use cases while maintaining simplicity and relevance.

III. METHODOLOGY

Figure 1 shows the overview of the dataset construction process. As the target programming language, we selected Java because it is widely used in various software projects and many projects are available on GitHub. The dataset construction process is divided into four steps: Java project collection, SBOM production with sbom-tool, automatic correction, and manual correction. First, we search Java projects on GitHub and clone their repositories. Next, we analyze the source code of the projects with an open-source SBOM generation tool: sbom-tool [8], and create initial SBOMs. After getting initial SBOMs, we correct missing information in the following steps. We automatically correct the SBOMs to fill in the essential fields with the information in the configuration files for Maven (pom.xml). Finally, we manually correct the SBOMs and create the SBOMs that follow the SPDX Lite profile.

A. Java Project Collection

We collected Java projects from GitHub. We obtained an initial list of Java projects using the GitHub API, by searching for repositories that met the following conditions: (1) whose primary language is Java; (2) that have at least 1000 stars; (3) the repository is not archived. These conditions were chosen to ensure that the projects are active and popular.

We obtained 3,271 GitHub repositories matching these conditions. We cloned these projects to analyze the source code of the projects in the next SBOM production step. 3,254 projects were successfully cloned and 17 projects failed due to some errors such as the path length limitation on the Windows environment.

B. SBOM Production with sbom-tool

We produced initial SBOMs for the cloned projects with sbom-tool [8], an open-source SBOM generation tool. This popular and actively maintained tool supports the generation of SBOMs for Java projects using Maven as a build tool.

First, we selected Java projects that use Maven as a build tool from the cloned projects. This is because sbom-tool does not support SBOM generation for Java projects using other build tools than Maven. We consider that a project uses Maven if it contains at least one pom.xml file, a configuration file for Maven. We gained 798 projects using Maven.

For these projects, we created SBOMs with sbom-tool. We used version 2.9.9 of sbom-tool, and ran it with the following command:

sbom-tool generate -b *output-dir* -bc *src-dir* -pn *name* -pv *version* -ps *supplier*

output-dir	directory path for the output SBOM.
src-dir	project root directory path.
name	project name (value: repository name).
version	project version (value: latest commit hash).
supplier	project supplier (value: repository owner name).

The primary language used in the collected projects is Java, but some projects contain code written in other languages. It means that they may have dependencies that are not managed by Maven. Sbom-tool can detect a package manager used in a project, so we checked the log output and excluded a project from the SBOM production candidates if sbom-tool detects package managers other than Maven.

C. Automatic Correction

SBOMs produced by sbom-tool do not contain values of some fields. Table I shows the essential fields about a package (e.g. library) in the SPDX Lite profile. The first column shows the essential SPDX fields, and the next two columns indicate whether information can be retrieved with SBOM production with sbom-tool and automatic correction steps respectively. In this table, "+" means that the field is filled in for all projects, while "o" means that the field is filled in for some projects because the data source is optional. As the column labeled "sbom-tool" shows, only the following fields are filled with sbom-tool: Package Name, Package SPDX Identifier, Package Version, Files Analyzed, and External Reference. Therefore, we need to fill in the other fields to create SBOMs that comply with the SPDX Lite profile.

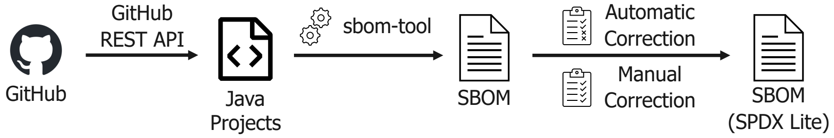

Fig. 1. Overview of the dataset construction process

TABLE I
ESSENTIAL FIELDS ABOUT PACKAGE

SPDX Field name	sbom-tool	Automatic Correction
Package Name	+	
Package SPDX Identifier	+	
Package Version	+	
Package File Name		+
Package Supplier		o
Package Download Location		+
Files Analyzed	+	
Package Home Page		o
Concluded License		o
Declared License		o
Comments on License		
Copyright Text		
Package Comment		
External Reference	+	

Most of the Java components are published on the Maven Central Repository[1]. It provides a REST API[2], and it enables us to obtain the information to fill some of the remaining fields. Table II shows the correspondence between fields and information that can be obtained using the API. We can get the file name of the jar file of a component (for the Package File Name field), and the URL to its jar file (for the Package Download Location field) through the API. The API also provides the URL to the configuration file of the component (pom.xml), which contains some additional information about the component. Using the information available on the Maven Central Repository, we can populate six fields for a component: *Package File Name, Package Supplier, Package Download Location, Package Home Page, Concluded License, and Declared License*. We wrote a script to obtain information from the API and fill in the fields. In this step, we automatically corrected the SBOMs by running the script.

In the Maven ecosystem, component names are case-sensitive. However, sbom-tool does not handle case sensitivity correctly, which occasionally causes the script to fail when attempting to fetch information, even if the component is available on the Maven Central Repository. Additionally, the script cannot retrieve information for components not published on the Maven Central Repository. To address these limitations, we excluded projects with such dependencies from the candidates for SBOM production. As a result, we successfully generated 46 SBOMs during the automatic correction step.

1) Package File Name & Package Download Location: The API provides the name of the jar file of a component and the URL to its jar file. We can use these two pieces of information as the values of the Package File Name field and the Package Download Location field, respectively.

TABLE II
INFORMATION AVAILABLE FROM MAVEN CENTRAL FOR AUTOMATIC CORRECTION

SPDX Field name	Information on Maven Central
Package File Name	Name of the jar file
Package Supplier	Organization or Developers (pom.xml)
Package Download Location	URL of the jar file
Package Home Page	URL of the homepage (pom.xml)
Concluded License	Licenses (pom.xml)
Declared License	Licenses (pom.xml)

2) Package Supplier: A developer can *optionally* include the organization to which a project belongs or the developer information of the component in the pom.xml. If the organization information is included in the pom.xml, we use it as the value of the Package Supplier field; otherwise, if a developer is included instead, the Package Supplier field is populated with that information. If multiple developers are included, we cannot automatically determine which developer to use as the value of the Package Supplier field, so the field is left blank.

3) Package Home Page: A developer can *optionally* include the URL to the project's homepage in pom.xml. If the information is included in pom.xml, we use it as the value of the Package Home Page field.

4) Concluded License & Declared License: A developer can *optionally* include the license information of the project itself in pom.xml. The license of the project is described using the name of the license and the URL of the license in pom.xml. In SPDX format, well-known licenses are represented by their SPDX License Identifiers (e.g. Apache License Version 2.0 is represented by Apache-2.0). If the name of the license or the URL of the license in pom.xml exactly matches that of a well-known license, use its SPDX License Identifier as the value of the Concluded License field and the Declared License field. There are some variations in the spelling of license names and URLs, the license information in pom.xml does not often match the well-known licenses exactly. In these cases, we use the information in pom.xml as is for the value of the two fields, and manually check and fix them in the next Manual Correction step.

D. Manual Correction

After the automatic correction step, there are still some fields whose values are empty or incorrect. During the manual correction step, we reviewed and corrected these fields to

[1] https://central.sonatype.com/

[2] https://central.sonatype.org/search/rest-api-guide/

ensure that the SBOMs complied with the SPDX Lite profile. Across the 46 SBOMs generated in the automatic correction step, a total of 1,752 fields were manually corrected.

To fill the fields, we manually collected information from the following sources:

1) pom.xml of the component
2) Web pages whose URL is written in the pom.xml
3) Web pages found by searching on the Internet

First, we checked the information written in pom.xml of the component. In the automatic correction step, our script sometimes failed to parse pom.xml correctly. We checked these pom.xml manually and filled the fields of the SBOMs.

Second, we utilized information from the webpages whose URL is listed in pom.xml. pom.xml contains the URL of the project's homepage, the URL of the issue tracker, and the URL of the source code repository (e.g. SourceForge, GitHub). We attempted to obtain the missing information from those web pages. If a URL is already broken, we checked the content of the web page using the Wayback Machine[3].

In most cases, we can fill in the fields using the information collected in the above two steps. However, in some cases, we could not find the information we needed from the pom.xml or the web pages. In such cases, we searched for the source code repository of the component on the Internet. If we found the source code repository, we extracted some information from it. If we could not find the information we needed from the above sources to fill a field, we filled it with "NOASSERTION", which is a special value defined in the SPDX specification to indicate that the information is not available.

After correcting all fields, we validated the SBOMs using the SPDX Online Tool[4], an official tool provided by SPDX, to ensure that they comply with the SPDX format.

IV. Dataset Structure

The structure of the dataset is shown in Table III. The dataset contains repositories.json, which is a list of Java projects. The list includes the project ID, the name of the project, the project URL, the Git commit hash used to generate the SBOM, the number of dependencies of the project, and the number of stars on GitHub. For each project, we created a directory named "[ID]_[Name (/ is replaced with +)]". It contains an SBOM file (sbom.spdx.json) and a file describing the data sources of the SBOM (sbom.data-sources.json).

The users of the dataset can select SBOM files based on the information in repositories.json and utilize them as input for SBOM consumption tools to evaluate their functionality. Additionally, detailed information about the data sources used to generate an SBOM is available in sbom.data-sources.json. The file includes the type of the source for each field value (auto or manual), the name of the source, and any associated URL. These details help users understand the process of SBOM creation and ensure transparency within the dataset.

TABLE III
Structure of the dataset

path	description
/repositories.json	The list of Java projects in the dataset
/[id]_[name]/	The directory for each Java project
|– sbom.spdx.json	The SBOM file for the project
|– sbom.data-sources.json	The file describing the data sources of the SBOM

V. Limitations

In the automatic correction step, we relied on information obtained from the component's pom.xml file to populate certain fields. However, during manual correction, we found that the pom.xml file occasionally contains inaccurate information. For example, if a project is a fork of another and its pom.xml file has not been updated, it may include incorrect details. Consequently, some values populated during the automatic correction step may contain errors.

Currently, the dataset comprises 46 SBOMs, which may be insufficient for a comprehensive evaluation of SBOM consumption tools. Nonetheless, the projects included in the dataset have complex dependency relationships, making it valuable for certain evaluation scenarios. In future work, we aim to expand the dataset by incorporating additional SBOMs for Java projects and extending it to SBOMs for projects written in other programming languages.

VI. Conclusion

This paper presents a dataset of SBOMs for evaluating SBOM consumption tools. It comprises 46 SBOMs generated from real-world Java projects collected from GitHub. The SBOMs were created using an open-source SBOM generation tool and refined through manual corrections to ensure accuracy. This dataset has room for improvement in terms of both its quality and quantity, so we plan to refine the data and expand it to include a broader range of projects across various programming languages.

Through the dataset construction process, we identified several issues with the SBOM generation tools. Sbom-tool does not properly handle case sensitivity in component names, potentially leading to false negatives in vulnerability detection by SBOM consumption tools. Additionally, creating an SBOM with all the necessary information currently requires significant manual effort. To facilitate effective dependency management using SBOMs, it is crucial to address these limitations and minimize the manual work needed to produce accurate and comprehensive SBOMs.

Acknowledgements

This work was supported by JSPS KAKENHI Grant Numbers JP24H00692, JP21K18302, JP21H04877, JP23K24823, JP22K11985, JP24K14895, JP21K02862, JP23K28065, and Nanzan University Pache Research Subsidy I-A-2 for the 2024 academic year.

[3] http://web.archive.org/

[4] https://tools.spdx.org/app/validate/

References

[1] M. Alfadel, D. E. Costa, and E. Shihab, "Empirical Analysis of Security Vulnerabilities in Python Packages," in *Proceedings of the 2021 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*, 2021, pp. 446–457.

[2] anchore, "grype," accessed on 5 February 2025. [Online]. Available: https://github.com/anchore/grype

[3] M. Balliu, B. Baudry, S. Bobadilla, M. Ekstedt, M. Monperrus, J. Ron, A. Sharma, G. Skoglund, C. Soto-Valero, and M. Wittlinger, "Challenges of Producing Software Bill of Materials for Java," *IEEE Security & Privacy*, pp. 2–13, 2023.

[4] Chainguard, "bom-shelter," 2022, accessed on 5 February 2025. [Online]. Available: https://github.com/chainguard-dev/bom-shelter

[5] CycloneDX, "bom-examples," accessed on 5 February 2025. [Online]. Available: https://github.com/CycloneDX/bom-examples

[6] A. Halbritter and D. Merli, "Accuracy evaluation of sbom tools for web applications and system-level software," in *Proceedings of the 19th International Conference on Availability, Reliability and Security*, ser. ARES '24. New York, NY, USA: Association for Computing Machinery, 2024. [Online]. Available: https://doi.org/10.1145/3664476.3670926

[7] R. G. Kula, D. M. German, A. Ouni, T. Ishio, and K. Inoue, "Do developers update their library dependencies?" *Empirical Software Engineering*, vol. 23, no. 1, pp. 384–417, Feb. 2018. [Online]. Available: https://doi.org/10.1007/s10664-017-9521-5

[8] Microsoft, "sbom-tool," accessed on 5 February 2025. [Online]. Available: https://github.com/microsoft/sbom-tool

[9] OWASP Foundation, "OWASP CycloneDX Software Bill of Materials (SBOM) Standard," accessed on 5 February 2025. [Online]. Available: https://cyclonedx.org/

[10] T. E. Parliament, "Cyber resilience act," 2024, accessed on 5 February 2025. [Online]. Available: https://www.europarl.europa.eu/doceo/document/TA-9-2024-0130_EN.html

[11] SPDX Workgroup, "About - Software Package Data Exchange (SPDX)," accessed on 5 February 2025. [Online]. Available: https://spdx.dev/about

[12] ——, "spdx-examples," accessed on 5 February 2025. [Online]. Available: https://github.com/spdx/spdx-examples

[13] ——, "Spdx lite," accessed on 5 February 2025. [Online]. Available: https://spdx.github.io/spdx-spec/v2.2.2/SPDX-Lite/

[14] T. Stalnaker, N. Wintersgill, O. Chaparro, M. Di Penta, D. M. German, and D. Poshyvanyk, "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)*, 2024, pp. 513–525. [Online]. Available: https://doi.org/10.1145/3597503.3623347

[15] The Linux Foundation, "The State of Software Bill of Materials (SBOM) and Cybersecurity Readiness," 2022, accessed on 5 February 2025. [Online]. Available: https://www.linuxfoundation.org/research/the-state-of-software-bill-of-materials-sbom-and-cybersecurity-readiness

[16] The White House, "Executive Order on Improving the Nation's Cybersecurity," 2021, accessed on 5 February 2025. [Online]. Available: https://www.federalregister.gov/d/2021-10460

[17] Y. Wang, B. Chen, K. Huang, B. Shi, C. Xu, X. Peng, Y. Wu, and Y. Liu, "An Empirical Study of Usages, Updates and Risks of Third-Party Libraries in Java Projects," in *Proceedings of the 2020 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, 2020, pp. 35–45.

[18] B. Xia, T. Bi, Z. Xing, Q. Lu, and L. Zhu, "An Empirical Study on Software Bill of Materials: Where We Stand and the Road Ahead," in *Proceedings of the IEEE/ACM 45th International Conference on Software Engineering (ICSE)*, 2023, pp. 2634–2646.