Osmy: A Tool for Periodic Software Vulnerability Assessment and File Integrity Verification using SPDX Documents

Rio Kishimoto*, Tetsuya Kanda*, Yuki Manabe†, Katsuro Inoue‡, Yoshiki Higo*
*Osaka University, Japan, {r-kisimt, t-kanda, higo}@ist.osaka-u.ac.jp
†The University of Fukuchiyama, Japan, manabe-yuki@fukuchiyama.ac.jp
‡Nanzan University, Japan, inoue599@nanzan-u.ac.jp

***Abstract*—Libraries have become integral to modern software development, yet their management often falls short, resulting in issues such as delayed responses to vulnerabilities. To address these issues, the use of a Software Bill of Materials (SBOM) is recommended. Despite the recommendation, there is a lack of tools supporting software management using SBOM. In this paper, we present "Osmy", a tool designed to facilitate effective software management using SBOM in the SPDX format—one of the major SBOM formats. Osmy is designed to simplify and streamline SBOM-based software management for end users. It automates vulnerability assessment and file integrity verification, operating periodically to ensure continuous protection. Users receive timely notification of any identified issues, ensuring a proactive approach to software security. Osmy is available at https://github.com/higolab/Osmy.**

***Index Terms*—SBOM, SPDX, vulnerability**

I. Introduction

In modern software development, developers use third-party libraries to efficiently reuse common functionalities between software projects. While the use of a library has the advantage of shortening development time and reducing development costs, it also has the disadvantage of incorporating defects contained in the library [10], [18]. Defects affecting the security of software are called vulnerabilities. When a vulnerability is discovered, it is necessary to quickly update software to a non-vulnerable version. However, delays and omissions in responding to vulnerabilities have become a problem because the management of third-party libraries used in the project is not enough [1], [7]. Taking the popular Java-based logging library Log4j as an example, a serious vulnerability was found in 2021 [4]. As of the November 2023 statistics, approximately 20% of Log4j downloads in the past month across multiple countries still comprised outdated versions, persisting with this identified vulnerability [11].

To address these problems, the use of a Software Bill of Materials (SBOM) is recommended [17]. An SBOM is a document that describes the elements such as packages and documentation that make up a particular piece of software, their licenses, and the relationships among the elements. SPDX [12] is one of the open standards for SBOM, and an SBOM described according to SPDX is called an SPDX document. Utilizing the information described in an SPDX document, we can scan vulnerabilities in a piece of software and the libraries where it depends. Additionally, we can verify the integrity of its files to ensure the validity of the vulnerability assessment based on the SPDX document. Thus, SPDX documents are valuable for effective software management.

However, despite the benefits of utilizing SPDX documents for software management, there is currently a lack of tools to support this process [20]. In particular, tools for software end users are lacking. In this paper, we present a tool called "Osmy", which makes vulnerability assessment and file integrity verification based on the information on SPDX documents easier and more labor-saving for software end users.

II. Osmy: Overview

Osmy is a tool that performs periodic vulnerability assessment and file integrity verification of software based on SPDX documents as shown in Fig. 1. Osmy consists of a server program and CLI/GUI clients. The server program has a list of software that is managed by Osmy using SPDX documents. It performs a vulnerability assessment and a file integrity verification periodically based on the information in SPDX documents. It also notifies users of those results by e-mail. The CLI/GUI clients provide a user interface to register/update SPDX documents that are used to perform vulnerability assessments and file integrity verifications. They also provide a user interface to check those results.

In the following, we explain the methods of vulnerability assessment and file integrity verification on the server side, and how to notify users of those results. Client-side functionality is described in Section III along with a usage scenario.

A. Vulnerability Assessment

Osmy performs vulnerability assessment in two steps: 1) identification of dependent packages, 2) retrieval of vulnerability information.

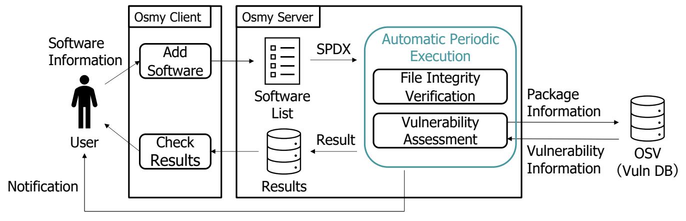


Fig. 1. Overview of Osmy

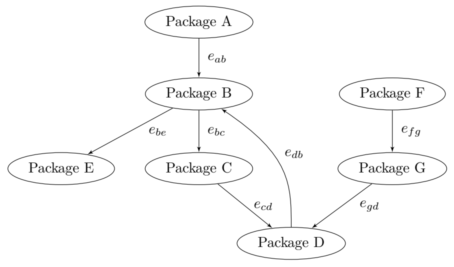


Fig. 2. Example of a dependency graph

1) Identification of dependent packages: Software dependencies consist of runtime dependencies and development-time dependencies (e.g. packages that only a testing process depends on). An SPDX document can include information about packages that are development-time dependencies as well as information about packages that are runtime dependencies. Vulnerabilities within runtime dependencies impact software vulnerabilities during runtime, whereas those within development-time dependencies do not. Therefore, Osmy identifies runtime dependencies of a piece of software by utilizing a dependency graph.

A dependency graph is a directed graph representing dependencies between packages. A vertex represents a package and a directed edge represents dependency between two packages. An example of a dependency graph is shown in Fig. 2. When one package directly or indirectly depends on another package, there is a path between two packages in a dependency graph. For example, in Fig. 2, there is a directed edge e_{ab} from Package A to Package B, and it means Package A directly depends on Package B. Also, since there is no path from Package A to Package G, Package A does not depend directly or indirectly on Package G.

Osmy creates a dependency graph from relationship information described in an SPDX document. Code. 1 shows an example of a part of an SPDX document in JSON format that describes relationships between elements. In this example, the relationships shown in Fig. 3 are described. In an SPDX document, a relationship is described using the type of relationship and the IDs of the two elements involved. The relationship described in lines 1 through 5 of the Code. 1 says a package (SPDXRef-RootPackage) depends (DEPENDS) on another package (SPDXRef-Package-log4net). Lines 6 through 10 indicate that the package (SPDXRef-Package-log4net) depends (DEPENDS) on another package (SPDXRef-Package-System.Xml.XmlDocument). Using this information, Osmy creates a dependency graph in two steps. First, it adds all packages whose information is described in an SPDX document as vertices of a graph. Next, it adds edges to the graph based on relationship information in the document. In this step, only relationships whose types indicate runtime dependencies are processed, and relationships whose types indicate development-time dependencies are ignored. The types of relationships that indicate runtime dependencies are shown in Table I, Table II, and Table III. Dependent packages are identified in the graph by a depth-first search from the software's own package (hereafter simply referred to as a root package). Packages visited during a search are identified as software's runtime dependencies because they have paths from a root package.

2) Retrieval of vulnerability information: For each identified package that is a runtime dependency, Osmy checks whether the package contains a vulnerability or not by querying the OSV database [5], which is a database that stores OSS vulnerabilities. The OSV database is queried via HTTP communication with the officially provided OSV API. Osmy fetches vulnerability information of packages by sending a request to the OSV batch execution API, which performs up to 1000 vulnerability information queries at once, with the name

Code. 1
EXAMPLE OF RELATIONSHIPS INFORMATION BETWEEN SPDX ELEMENTS

```
 1  {
 2    "relationshipType": "DEPENDS_ON",
 3    "spdxElementId": "SPDXRef-RootPackage",
 4    "relatedSpdxElement": "SPDXRef-Package-log4net"
 5  },
 6  {
 7    "relationshipType": "DEPENDS_ON",
 8    "spdxElementId": "SPDXRef-Package-log4net",
 9    "relatedSpdxElement": "SPDXRef-Package-System.Xml.Xml
          Document"
10  }
```

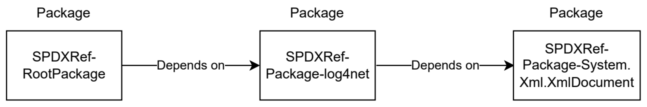


Fig. 3. Relationship described in Code. 1

TABLE I
RELATIONSHIPS THAT A MAY INHERIT A VULNERABILITY THAT B CONTAINS

Relationship	Description
CONTAINS	A contains B
DYNAMIC_LINK	A dynamically links to B
EXPANDED_FROM_ARCHIVE	A is expanded from the archive B
FILE_ADDED	A is a file that was added to B
GENERATED_FROM	A was generated from B
PATCH_FOR	A is a patch file for B
STATIC_LINK	A statically links to B
HAS_PREREQUISITE	A has as a prerequisite B
DEPENDS_ON	A depends on B

TABLE II
RELATIONSHIPS THAT B MAY INHERIT A VULNERABILITY THAT A CONTAINS

Relationship	Description
CONTAINED_BY	A is contained by B
DISTRIBUTION_ARTIFACT	distributing A requires that B also be distributed
GENERATES	A generates B
OPTIONAL_COMPONENT_OF	A is an optional component of B
PATCH_APPLIED	A is a patch file that has been applied to B
PREREQUISITE_FOR	A is a prerequisite for B
DEPENDENCY_OF	A is a dependency of B
OPTIONAL_DEPENDENCY_OF	A is an optional dependency of B
RUNTIME_DEPENDENCY_OF	A is a dependency required for the execution of B

TABLE III
RELATIONSHIPS THAT IF ONE IS VULNERABLE, THEN THE OTHER MAY ALSO BE VULNERABLE

Relationship	Description
COPY_OF	A is an exact copy of B
PACKAGE_OF	A is used as a package as part of B
VARIANT_OF	A is a variant of B

and version of the packages. In addition, since popular libraries appear in multiple SPDX documents, queries for packages with the same name and version are combined into a single query to reduce the number of queries.

B. File Integrity Verification

Osmy checks for tampering and corruption in software files by verifying the checksum of each file. This ensures the content of an SPDX document matches the information of the corresponding installed software.

Osmy verifies the checksum of each file using the information described in a file information section of an SPDX document. This section lists the files composing a piece of software, along with the name and checksum for each file. Code. 2 shows an example of the section in an SPDX document in JSON format. This example shows the information of a file whose name is log4net.dll, and the checksum calculated with SHA-1 is written. The SPDX format permits the inclusion of multiple hash values, computed using different algorithms, as the checksum of a file. Specifically, the SHA-1 hash value must be specified. Osmy verifies the integrity of a file by comparing the SHA-1 hash value recorded in the file information of an SPDX document with the one calculated from the actual file on the system. A mismatch indicates potential tampering or corruption.

Code. 2
EXAMPLE OF FILE INFORMATION

```
{
  "fileName": "./log4net.dll",
  "SPDXID": "SPDXRef-File--log4net.dll",
  "checksums": [
    {
      "algorithm": "SHA1",
      "checksumValue": "40fdba136f864c8a2f3e3f9c9e3949
           f7582a6077"
    }
  ],
  "licenseConcluded": "NOASSERTION",
  "licenseInfoInFiles": [ "NOASSERTION" ],
  "copyrightText": "NOASSERTION"
}
```

C. Notification to Users

Osmy conducts periodic vulnerability assessments and file integrity verifications, saving the results in a database. Users can customize the execution intervals for these processes in Osmy's configuration file. If the user configures e-mail notification settings, Osmy sends results to the user by e-mail when a vulnerability or file integrity error is detected. The e-mail message provides a summary of the vulnerability assessment or the file integrity verification. When Osmy detects a vulnerability, the message includes the number and names of vulnerable software, and when it detects a file integrity error, the message includes the number and names of software with file integrity errors. By notifying users when a problem is found, Osmy helps the user to respond to the problem as soon as possible.

III. ILLUSTRATIVE USAGE SCENARIO

In this section, we present a usage instance of Osmy to detect vulnerabilities and file integrity errors in software. Additionally, we describe the functionality of Osmy's client programs along with a usage scenario. As detailed in Section II, Osmy has two client programs: a GUI client and a CLI client. Users have the flexibility to utilize either or both to add software for management within Osmy. In this scenario, the user uses the GUI client.

A. Usage Scenario

First, the user configures Osmy. The user sets automatic execution intervals of vulnerability assessment and file integrity verification and e-mail notification settings in the configuration file of Osmy. After configuration, the user starts the server program of Osmy.

Next, the user adds software to the list of managed software. The user uses a client program of Osmy to add software to the list. The GUI client is shown in Fig. 4. The user can add software to the list by clicking the "Add" button in the lower-left corner of the window. When the user clicks the "Add" button, a dialog to enter software information is displayed (Fig. 5). The user enters a display name of the software, a

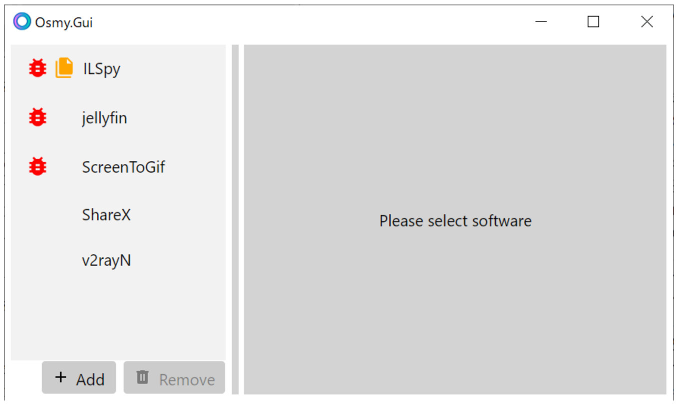
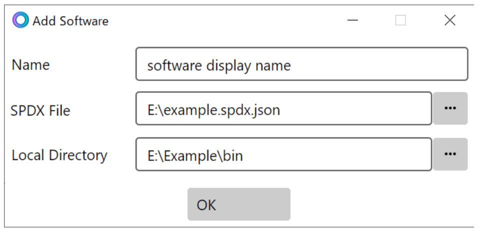


Fig. 4. GUI client



Name	software display name
SPDX File	E:\example.spdx.json
Local Directory	E:\Example\bin

Fig. 5. Dialog to add software

path to an SPDX document of the software, and a path to the directory where the software is located in the dialog. After filling in the dialog, the user clicks the "OK" button to add the software to the list. In this use case, the user adds five software (ILSpy[1], jellyfin[2], ScreenToGif[3], ShareX[4], v2rayN[5]). Added software is displayed in the software list on the left side of the window, and Osmy executes initial vulnerability assessment and file integrity verification for them.

B. Detected Vulnerabilities and File Integrity Errors

In this section, we illustrate some instances of detected vulnerabilities and file integrity errors. Fig. 4 shows the client when Osmy manages the five software. On the left side of the window, the list of software displays a red bug icon next to the name of vulnerable software. Similarly, if a piece of software has file integrity errors, a yellow file icon is shown next to its name. In this case, the user can see that three software contain vulnerabilities and one software has file integrity errors. By selecting a piece of software in the list, the user can check detailed information about it.

1) Vulnerabilities: Fig. 6 shows the client after selecting the software with vulnerabilities. In the "Packages" tab, packages containing vulnerabilities are highlighted in red in the list of packages. In this example, one of the dependent packages of ScreenToGif: System.Data.SqlClient with version 4.8.3 contains a vulnerability whose IDs are GHSA-8g2p-5pqh-5jmc and CVE-2022-41064.

2) File Integrity Errors: Fig. 7 shows the client after selecting the software with file integrity errors. In the "Files" tab, the status of a file is indicated by an icon next to the file name. The green file icon means that the file checksum matches the legitimate checksum written in the SPDX document. The yellow file icon means that the file checksum does not match it, and the dotted file icon means that the file does not exist nevertheless the SPDX document contains its information. In this example, ILSpy.dll is corrupted and AvalonDock.dll is missing.

IV. Related Work

OSV-Scanner [6] and spdx-to-osv [13] are command line tools to perform vulnerability assessment based on information in SPDX documents, which can be easily incorporated into CI/CD pipelines. Dependency-Track [14] is a software composition analysis tool, which leverages SBOMs written in CycloneDX [9] format. These tools are designed for software developers to manage dependent packages of their software projects in their development process. In this paper, we described Osmy, which is designed for software end users and provides features for them such as GUI, periodic execution, and notifications. SBOM's states of practice and challenges have been investigated in [3], [8], [15], [16], [20], and Balliu et al. have investigated the maturity of SBOM producers for Java projects in [2]. Wermke et al. have discussed security challenges introduced by adopting open-source components [19].

V. Summary

In this paper, we presented "Osmy", which automatically performs software vulnerability assessment and file integrity verification using SPDX documents periodically. We demonstrated the use of Osmy to detect vulnerabilities and file integrity errors in software. We have future work to assess the effectiveness of Osmy by evaluating its performance and usability when managing a larger number of SPDX documents and to improve the user interface to help users prioritize vulnerabilities.

Acknowledgements

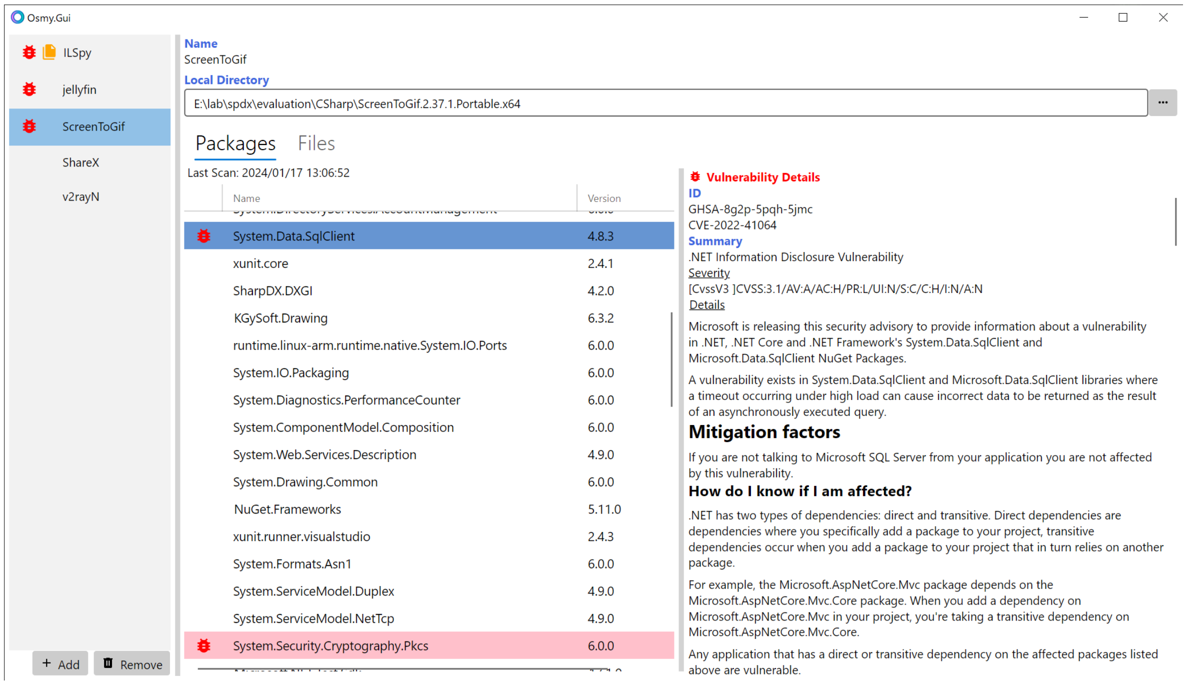
This work was supported by JSPS KAKENHI Grant Numbers JP23H03375, JP21K02862, JP19K20239, JP20H04166, JP21K18302, JP21K11829, JP21H04877, JP22H03567, JP22K11985, and Nanzan University Pache Research Subsidy I-A-2 for the 2023 academic year.

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] 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.

[1]https://github.com/icsharpcode/ILSpy/tree/2ef324515
[2]https://github.com/jellyfin/jellyfin/tree/cc3d087
[3]https://github.com/NickeManarin/ScreenToGif/tree/867c5f6
[4]https://github.com/ShareX/ShareX/tree/d491b1f
[5]https://github.com/2dust/v2rayN/tree/5ceb638



Name
ScreenToGif
Local Directory
E:\lab\spdx\evaluation\CSharp\ScreenToGif.2.37.1.Portable.x64

Last Scan: 2024/01/17 13:06:52

Name	Version
System.Data.SqlClient	4.8.3
xunit.core	2.4.1
SharpDX.DXGI	4.2.0
KGySoft.Drawing	6.3.2
runtime.linux-arm.runtime.native.System.IO.Ports	6.0.0
System.IO.Packaging	6.0.0
System.Diagnostics.PerformanceCounter	6.0.0
System.ComponentModel.Composition	6.0.0
System.Web.Services.Description	4.9.0
System.Drawing.Common	6.0.0
NuGet.Frameworks	5.11.0
xunit.runner.visualstudio	2.4.3
System.Formats.Asn1	6.0.0
System.ServiceModel.Duplex	4.9.0
System.ServiceModel.NetTcp	4.9.0
System.Security.Cryptography.Pkcs	6.0.0

Vulnerability Details
ID
GHSA-8g2p-5pqh-5jmc
CVE-2022-41064
Summary
.NET Information Disclosure Vulnerability
Severity
[CvssV3]CVSS:3.1/AV:A/AC:H/PR:L/UI:N/S:C/C:H/I:N/A:N
Details

Microsoft is releasing this security advisory to provide information about a vulnerability in .NET, .NET Core and .NET Framework's System.Data.SqlClient and Microsoft.Data.SqlClient NuGet Packages.

A vulnerability exists in System.Data.SqlClient and Microsoft.Data.SqlClient libraries where a timeout occurring under high load can cause incorrect data to be returned as the result of an asynchronously executed query.

Mitigation factors

If you are not talking to Microsoft SQL Server from your application you are not affected by this vulnerability.

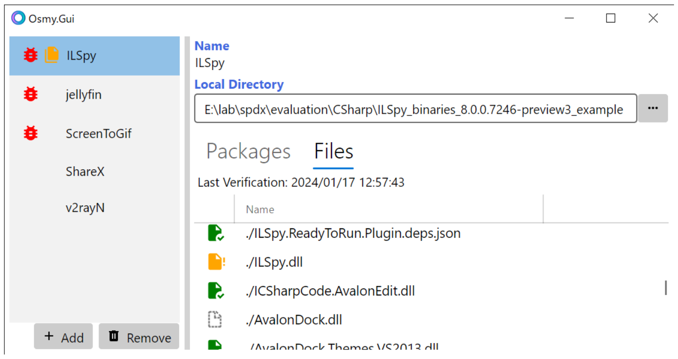
How do I know if I am affected?

.NET has two types of dependencies: direct and transitive. Direct dependencies are dependencies where you specifically add a package to your project, transitive dependencies occur when you add a package to your project that in turn relies on another package.

For example, the Microsoft.AspNetCore.Mvc package depends on the Microsoft.AspNetCore.Mvc.Core package. When you add a dependency on Microsoft.AspNetCore.Mvc in your project, you're taking a transitive dependency on Microsoft.AspNetCore.Mvc.Core.

Any application that has a direct or transitive dependency on the affected packages listed above are vulnerable.

Fig. 6. Packages Tab



Name
ILSpy
Local Directory
E:\lab\spdx\evaluation\CSharp\ILSpy_binaries_8.0.0.7246-preview3_example

Last Verification: 2024/01/17 12:57:43

Name
./ILSpy.ReadyToRun.Plugin.deps.json
./ILSpy.dll
./ICSharpCode.AvalonEdit.dll
./AvalonDock.dll

Fig. 7. Files Tab

[3] T. Bi, B. Xia, Z. Xing, Q. Lu, and L. Zhu, "On the Way to SBOMs: Investigating Design Issues and Solutions in Practice," 2023, arXiv:2304.13261.

[4] CISA, "Mitigating Log4Shell and Other Log4j-Related Vulnerabilities," 2021, accessed on 4 November 2023. [Online]. Available: https://www.cisa.gov/news-events/cybersecurity-advisories/aa21-356a

[5] Google, "OSV: Open source vulnerability DB and triage service," https://osv.dev/.

[6] ——, "OSV-Scanner," https://github.com/google/osv-scanner, accessed on 4 November 2023.

[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] S. Nocera, S. Romano, M. D. Penta, R. Francese, and G. Scanniello, "Software Bill of Materials Adoption: A Mining Study from GitHub," in *Proceedings of the 2023 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, 2023, pp. 39–49.

[9] OWASP Foundation, "OWASP CycloneDX Software Bill of Materials (SBOM) Standard," https://cyclonedx.org/.

[10] O. P. N. Slyngstad, A. Gupta, R. Conradi, P. Mohagheghi, H. Rønneberg, and E. Landre, "An Empirical Study of Developers Views on Software Reuse in Statoil ASA," in *Proceedings of the 2006 ACM/IEEE International Symposium on Empirical Software Engineering (ISESE)*, 2006, pp. 242–251.

[11] Sonatype, "Log4j Updates and Vulnerability Resources," accessed on 4 November 2023. [Online]. Available: https://www.sonatype.com/resources/log4j-vulnerability-resource-center

[12] SPDX Workgroup, "About - Software Package Data Exchange (SPDX)," https://spdx.dev/about.

[13] ——, "spdx-to-osv: Produce an Open Source Vulnerability JSON file based on information in an SPDX document," https://github.com/spdx/spdx-to-osv, accessed on 4 November 2023.

[14] S. Springett, "Dependency-Track: an intelligent Component Analysis platform that allows organizations to identify and reduce risk in the software supply chain," https://github.com/DependencyTrack/dependency-track.

[15] T. Stalnaker, N. Wintersgill, O. Chaparro, M. D. Penta, D. M. German, and D. Poshyvanyk, "BOMs Away! Inside the Minds of Stakeholders: A Comprehensive Study of Bills of Materials for Software Systems," 2023, accepted to ICSE2024.

[16] The Linux Foundation, "The State of Software Bill of Materials (SBOM) and Cybersecurity Readiness," 2022, accessed on 4 November 2023. [Online]. Available: https://www.linuxfoundation.org/research/the-state-of-software-bill-of-materials-sbom-and-cybersecurity-readiness

[17] The White House, "Executive Order on Improving the Nation's Cybersecurity," 2021, accessed on 4 November 2023. [Online]. Available: https://www.whitehouse.gov/briefing-room/presidential-actions/2021/05/12/executive-order-on-improving-the-nations-cybersecurity/

[18] 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.

[19] D. Wermke, J. H. Klemmer, N. Wöhler, J. Schmüser, H. S. Ramulu, Y. Acar, and S. Fahl, ""Always Contribute Back": A Qualitative Study on Security Challenges of the Open Source Supply Chain," in *2023 IEEE Symposium on Security and Privacy (SP)*, 2023, pp. 1545–1560.

[20] 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 2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*, 2023, pp. 2634–2646.