2023 IEEE International Conference on Software Maintenance and Evolution (ICSME) | 979-8-3503-2783-0/23/$31.00 ©2023 IEEE | DOI: 10.1109/ICSME58846.2023.00016

Software Bill of Materials Adoption: A Mining Study from GitHub

Sabato Nocera*, Simone Romano*, Massimiliano Di Penta+, Rita Francese*, and Giuseppe Scanniello*
*University of Salerno, Fisciano, Italy
+University of Sannio, Benevento, Italy
email:{snocera, siromano}@unisa.it; dipenta@unisannio.it; {francese, gscanniello}@unisa.it

***Abstract*—A *Software Bill of Materials* (*SBOM*) is a complete, formally structured list of all the open-source and proprietary software components present in a software product, including their licenses, versions, and vendors. SBOMs enable software creators and consumers to gain visibility into the software supply chain and monitor any risks associated with security or licensing. Thereby, the United States Government and the European Union have brought SBOMs to the forefront of digital policy. In this paper, we present the results of an exploratory mining study that aims to investigate the adoption of SBOMs by open-source software projects. To that end, we mined GitHub to identify repositories that use SBOM generation tools developed by *SPDX* and *CycloneDX*, identifying a total of 186 public repositories adopting SBOMs. We found that the adoption of SBOMs is low, yet it has an increasing trend. Moreover, SBOM files are available in the repository or published release versions in 46% of the software projects analyzed. SBOMs are getting an increasing attention from software creators and consumers. There is a pressing need for organizations to update their software to meet the new standards required for the software supply chain.**

***Index Terms*—SBOM, Bill of Materials, Software Supply Chain**

I. Introduction

A *Software Bill of Materials* (*SBOM*) is a complete, formally structured list of all the open-source and proprietary software components (*e.g.,* libraries, frameworks, *etc.*) present in a software product (*e.g.,* a software application), including their licenses, versions, and vendors. SBOMs are analogous to *Bill of Materials* (*BOMs*) in supply chains and manufacturing. BOMs are used by manufacturers to keep track of the components utilized to make a product. Whenever a defect is identified in a component, a BOM makes it simple to identify the affected product and solve the issue, possibly alerting the users. SBOMs apply the same approach to software products, allowing developers to identify, trace, and address not only reliability- and security-associated risks, but also legal ones, originating from the integration of software with incompatible licenses.

The need for developing a mechanism such as SBOM has originated, on the one hand, from the peculiar nature of modern software, which no longer consists of proprietary software only. That is, the software is usually developed leveraging existing closed-and open-source components, resulting in complex supply chains, which are only as secure as their least secure component. The *OWASP Top 10*, one of the most trustworthy sources for rating software security concerns, identifies the usage of outdated and vulnerable components as one of the most critical security risks [21]. For instance, the *Heartbleed* vulnerability in the *OpenSSL* cryptography library, disclosed in 2014, has cost businesses at least 500 million USD [6]. At the same time, public administrations are raising the need for secure and accountable software. In 2021, United States President Biden issued an executive order outlining how federal departments, agencies, and contractors doing business with the government must safeguard their software in response to supply chain security risks, including making use of SBOMs [10]. Although the executive order is directed toward organizations doing business with the government, these guidelines, including SBOMs, are expected to shortly become the *de facto* standard for how all organizations develop and maintain their software projects. In summary, while the use of SBOMs has been motivated by security-related organizations and, basically, enforced by public administrations, its adoption is still limited, as pointed out by recent studies [14], [33], [34].

In this paper, we complement the existing empirical literature on the adoption of SBOMs by analyzing the extent to which SBOMs are used in existing open-source projects. Rather than surveying and interviewing developers (which has been done by previous literature), we analyze the extent to which open-source projects leverage SBOM generation tools and publish SBOMs in their repositories or public release versions. More specifically, we identified 186 public repositories on GitHub that generate their own SBOM in one of the two most widely used SBOM standards, namely *SPDX* and *CycloneDX* [33]. Each repository generates its SBOM using at least one SBOM generation tool among 19 different ones owned by SPDX or CycloneDX.

We observed that, while SBOM adoption in the analyzed open-source repositories (repositories and projects are terms we use interchangeably in this paper) is still pretty low, it is trending upward among both software creators and consumers. This may be a result of increased pressure and interest from major players, such as the United States Government. At the same time, we found that in 46% of the software projects examined, SBOM files are accessible in the repository or public release versions. There exists an urgent necessity for organizations to upgrade their software to comply with the latest standards mandated for the software supply chain.

Among the others, the main contributions of this paper can

2576-3148/23/$31.00 ©2023 IEEE
DOI 10.1109/ICSME58846.2023.00016

be summarized as follows:

1) We conducted the first study analyzing, from a quantitative perspective, the adoption of SBOM tools and the availability of SBOMs in open-source repositories or public release versions. To some extent, our study fills the gap of past qualitative investigations on SBOM usage [14], [33], [34] in the context of open-source projects hosted on GitHub.
2) We provide a curated dataset [17] about open-source projects depending on tools for SBOM generation, and information about their SBOM usage. We made available this dataset for both replication purposes and to aid research on SBOMs.

Paper Structure. In Section II, we provide some background information about SBOMs and discuss related work. In Section III, we present the study design, while the obtained results are reported in Section IV. A discussion of the main findings is reported in Section V, along with possible threats to their validity. We conclude the paper in Section VI.

II. Backgrounds and Related Work

In this section, we first provide some background notions about software supply chains and SBOMs. Then, we discuss related literature about SBOMs, software supply chains, and software dependencies.

A. Software Supply Chains and Software Bills of Materials

A software supply chain consists of anything needed (*e.g.,* software components and processes) to develop and deliver a software product. An SBOM describes the inventory of components used in a software product, which is often made up of both open-source and commercial software components [30]. An SBOM is useful for software creators and consumers. Software creators often leverage available third-party open- and closed-source software components, therefore an SBOM allows them to make sure those components are up-to-date and to check for known software vulnerabilities—*i.e.,* software security issues that would negatively affect its confidentiality, integrity, and/or availability [24]. Software supply chain attacks involve the injection of malicious code into a software package with the intention to compromise directly and indirectly dependent software systems [20]. By leveraging a more mature and informed risk-based process, software creators are able to more proactively evaluate and remediate security risks in the software supply chain, thereby improving customers' and users' trust.

Software supply chain and its security have been recognized as a top priority by both the United States Government—*i.e.,* President Biden's *Executive Order on Improving the Nation's Cybersecurity* [10]—and the European Union—*i.e.,* in the *Guidelines for Securing the Internet of Things* by *ENISA* (*European Union Agency for Cybersecurity*) [5]. Indeed, SBOMs have received notable policy attention from the United States Government, but the software industry has yet to implement them fully [34].

The description of the component dependencies of a software product through SBOMs should follow a standard convention to be consulted organically by software creators, consumers, and end-users. SBOM generation must produce a comprehensive and accurate list of all dependencies, which might vary depending on the stage of software development (*e.g.,* considering pre-build versus post-build artifacts) [16]. Manual SBOM generation is impractical due to the large number of dependencies that a software product can have. Therefore, it is necessary to rely on automated tools.

Different standards have been developed with the aim of providing a common structure to SBOMs. An SBOM standard is a machine-readable schema conceived to provide a uniform structure for describing the composition of software products so that it can be processed by other tools (*e.g.,* vulnerability or license scanners). SBOMs should comprise all the essential details required to depict their supply chain and utilization in information-technology operations while maintaining simplicity and conciseness [15]. As pointed out by Xia *et al.* [33], the most commonly adopted standards are SPDX (Software Product Data Exchange) [29] and CycloneDX [22].

SPDX is an open-source machine-readable format with origins in *Linux Foundation*. It was recently approved as ISO/IEC standard [19]. SPDX is designed to facilitate ingestion within a developer workflow and within businesses to support software compliance and transparency for open-source and proprietary code. The open nature of the standard, as well as the availability of software tools to generate it, should ease its adoption. Our investigation goes in the direction to understand if this assumption holds in open-source projects hosted on GitHub. CycloneDX is an open-source format for SBOM as well. It is an OWASP lightweight standard and specifically designed for use in software security contexts and supply chain component analysis [19]. It can communicate the inventory of software components, external services, and their relationships with one another. CycloneDX incorporates existing specifications including *Package URL*, *CPE*, and *SWID*. As for SWID, its tags are considered more of a software identifier than an SBOM format. Although SWID provides a simple and transparent way to track software inventory, only the SPDX and CycloneDX formats are recognized officially as an SBOM standard format. This was why we focused only on them.

B. Empirical Work on Software Bills of Material

Although a few secondary empirical studies (*e.g.,* surveys and gray literature reviews) have been conducted in the past, little investigation has been performed for what concerns the SBOM adoption.

The Linux Foundation's SBOM readiness survey in 2022 [14] on 412 worldwide organizations showed gaps in familiarity with, production planning for, and consumption of SBOMs. For example, 40% of the organizations were unclear about whether the industry is committed to mandating SBOMs, raising concerns about whether the software industry is abiding by President Biden's executive order governing

SBOMs. Our aim is to understand if the open-source community is adopting SBOM and to what extent.

Recently, Xia *et al.* [33] conducted 17 interviews and a survey with 65 respondents to investigate *(i)* the adoption of SBOM in practice, *(ii)* the SBOM tool support, and *(iii)* the main concerns related to the use of SBOMs. Based on the collected results, the authors derived 26 statements describing the SBOM adoption and its challenges. Those are related, for example, to the still limited awareness, the need for better incentives in using SBOMs, a limited consensus on what should be the content of SBOMs, and the presence of sub-optimal and non-standardized tools, especially for what concerns the support for handling vulnerabilities. We have looked at the adoption of SBOMs from a different perspective than Xia *et al.* [33]—*i.e.,* looking at the SBOM tool usage and SBOM files in open-source software projects rather than interviewing and surveying developers. Therefore, our study is complementary to that by Xia *et al.* [33], and together provide a better picture of the adoption of SBOMs.

C. Empirical Studies on Software Supply Chains

Other relevant literature is not directly related to SBOM, yet they study the challenges and risks arising from software supply chains. These risks can be, indeed, mitigated through the adoption of SBOMs.

Levy [13] asserts that open-source software can be more difficult to secure than proprietary software because of its development environment, which is more complex to a large extent. For example, organizations like Linux package the software commonly accepting code from others and bundle it with other packages. The resulting software is then distributed through websites. This complex web of interrelated parties creates several attack points in the software supply chain. The author highlights the need for technologies to secure supply chains of open-source software.

To enable sharing among industry practitioners having practical experiences and challenges with software supply chain security, Enck and Williams [4] conduct two industry and one government software summits (for a total of 30 organizations across the three summits). The result of these summits is a list of five challenges in software supply chain security. Among these five top challenges, there is *leveraging SBOM for security*. While participants had widely divergent opinions, they agree that efforts at establishing standards and requirements for SBOMs have the potential to lay the groundwork for innovative security enhancements.

Hartmann and Trew [8] propose the concept of *Context Variability* to model multiple product lines and to support software supply chains. In detail, this model can be used to facilitate the process of merging feature models during staged configuration where variability is originated from different product types, geographic regions, and customers. Preliminary experiments have been conducted to assess the proposed model.

Lamb and Zacchiroli [12] examine Reproducible-Builds (R-B) from the perspective of software professionals. R-B is an approach that can determine whether generated binaries correspond to the original source code. The main idea is that if we are confident that building a given source tree always generates bit-for-bit identical results, we can trust these artifacts. The results of their study indicate that uncontrolled build inputs occur when tool chains allow the build process to be affected by the surrounding environment. Similarly to Lamb and Zacchiroli [12], Vu *et al.* [31] propose an approach, motivated by the intuition behind R-B, to detect injected code in typosquatting and hijacked packages. In particular, their approach detects code injected into software packages by comparing their distributed artifacts (*PyPI*) with the source code repository (GitHub). The approach has been preliminary evaluated on two datasets of known malicious packages and the top ten most downloaded packages. Such an approach captures the known malicious packages and reduces the review effort for distributed software releases by 97%.

D. Empirical Studies on Software Project Inter-Dependencies

There are also studies related to the analysis of dependencies in the software ecosystem and their evolution. For example, Bavota *et al.* [1] observe that the developers of the Apache Java ecosystem try to lower the impact that dependency upgrades would have when triggered by major releases and critical bug fixes. As for dependencies in R and npm ecosystems, Bogart *et al.* [2] show that developers do not use systematic approaches to cope with changes. Later, Zahan *et al.* [35] analyzed the metadata of 1.63 million JavaScript npm packages and proposed six alerts of security weaknesses (*e.g.,* install scripts and maintainer accounts associated with an expired email domain) in a software supply chain. Zahan *et al.* identified 11 malicious packages from the install scripts signal and have also found 2,818 maintainer email addresses associated with expired domains (allowing the hijacking of 8,494 packages). The authors also obtain qualitative feedback on link signals through a survey responded to by 470 npm package developers. One of the most important results of the survey is that the developers would want to be notified about weak link signals before using third-party packages.

III. Study Design

The **goal** of our study is to analyze the adoption of SBOMs on open-source projects with the **purpose** of understanding *(i)* to what extent SBOMs are adopted; *(ii)* what is the adoption trend of SBOMs; and *(iii)* how SBOMs are managed in terms of where SBOM files are placed and the frequency with which these files are updated. The **perspective** is of researchers interested in acquiring information on the adoption of SBOMs. The **context** is represented by open-source software repositories/projects publicly available on GitHub and SBOMs that are compliant with the SPDX and CycloneDX standards.

A. Research Questions

Based on the aforementioned study goal, we devised four Research Questions (RQs).

RQ1. *Are SBOMs adopted by open-source software projects?* The aim of this RQ is to understand whether open-source software projects adopt SBOMs by looking at

those projects that use SBOM generation tools related to the SPDX and CycloneDX standards.

RQ2. *What is the adoption trend of SBOMs by open-source software projects and their owners?* To answer this RQ we study the adoption trend of SBOMs in relation to certain events of interest (*e.g.,* President Biden's executive order) that might increase the adoption of SBOMs by both open-source software projects and their owners.

RQ3. *How do open-source projects release SBOM files?* With this RQ, we start investigating how SBOMs are managed and, in particular, how they are distributed in open-source software projects, namely: where SBOM files are placed (*e.g.,* under version control or just in release versions).

RQ4. *How frequently are SBOM files updated within open-source software projects?* This last RQ contributes to understanding how SBOMs are managed in open-source software projects and, specifically, the frequency with which developers update the SBOM files made available in these projects.

B. Study Context and Planning

The context of our study consists of open-source software projects whose software repositories are publicly available on GitHub—which is currently the most popular version control platform for both personal and professional use [27]—and that could adopt GitHub-traceable SBOM generation tools related to the SPDX or CycloneDX standards. In other words, we considered as part of the context those software repositories that adopt SBOM generation tools developed by SPDX or CycloneDX and whose programming language is supported by the *GitHub Dependency Graph* [7]—a tool provided by GitHub to identify all dependencies of a software repository.

General information regarding the repositories (*e.g.,* number of commits and stars) and their owners (*e.g.,* provenance) was collected through *PyGithub* [23], a Python library that leverages *GitHub API v3*. As for commit-level information (*e.g.,* file modification dates and messages), it was retrieved through *PyDriller* [26].

The analysis of data retrieved from GitHub may lead to conclusions that are not representative of the context that was intended to be investigated in our study [11]. To deal with this issue, we excluded repositories that:

- Are archived, and whose last push date was more than one month before the query date (March 5, 2023). The rationale is to avoid selecting inactive software projects.
- Were forks. We made this action to avoid selecting duplicate software projects.
- Have a number of commits less than 100. The rationale here is to avoid selecting software projects with a too-short commit history.
- Are not related to real open-source software projects. This is to discard tutorials, books, students' assignments, and researchers' papers. To do so, we searched within the name of the repositories and their owners for any instances of the terms `paper`, `book`, `tutorial`, `learn`, `bootcamp`, `sample`, `example`, `demo`, `test`, `mirror`, `copy`, `community`, `docs`, `theme`, and `template`. Then, we manually checked the suitability of the matched repositories and decided whether or not to discard them.

C. Data Analysis

In this section, we detail how data analysis has been performed to answer the defined RQs. In particular, we show the associated data analysis conducted for each RQ.

1) RQ1: Are SBOMs adopted by open-source software projects?: We leveraged the GitHub Dependency Graph, which automatically detects the dependencies of a GitHub project on the basis of the content of its manifest and locks files available in the default branch of the software repository or by looking at the dependencies explicitly stated through the *GitHub Dependency Submission API*. For each repository, the dependency graph of GitHub shows: *(i)* the *dependencies—i.e.,* the ecosystems and packages it depends on—and *(ii)* the *dependents—i.e.,* the repositories and packages that depend on it.

Note that, while in principle any project could explicitly state its dependencies through GitHub Dependency Submission API, this API is still released as a beta version, and the Dependency Graph feature is in practice employed only according to the default settings. That is, when a software project employs a supported package manager, its dependencies are automatically mined by analyzing the manifest and lock files (*e.g.,* `requirements.txt` for PyPi or `pom.xml` for *Maven*). The complete list of all package managers supported by the GitHub Dependency Graph is available in its official documentation [7].

To identify the software projects that adopted SBOM:

1) We selected SBOM generation tools among those whose repository was available on GitHub and whose owners were SPDX or CycloneDX;
2) For each of the selected SBOM generation tools, we retrieved the list of repositories *dependent* on them by using the GitHub Dependency Graph—this list of repositories is that of open-source software projects that have adopted SBOMs.

It is important to note that SBOM generation tools can be integrated and used within the software development pipeline not only to produce SBOMs but they could also be used to consume other SBOMs. Therefore, we needed to identify which software projects make use of these tools to generate their SBOMs. We relied on the classification of the SBOM generation tools provided by their owners, *i.e.,* SPDX [25] and CycloneDX [3]. Accordingly, an SBOM generation tool can be:

- Integrated within build systems or package managers, thus SBOM files are automatically created when building a software artifact;
- Used as part of a Continuous Integration and Continuous Delivery (CI/CD) workflow/pipeline (*e.g., GitHub Action*);

- Used as modular components (*i.e.,* libraries) that can programmatically produce, consume, or transform SBOMs.

We assumed that software projects using SBOM generation tools integrated into building systems (*e.g.,* Maven) or CI/CD workflows (*e.g.,* GitHub Actions) produce their SBOMs. In all other cases, to ensure the actual adoption of SBOM, we looked for the presence of an SBOM file in the repository or in the latest release and excluded those projects that did not have any. To do this, we cloned each repository locally and downloaded all the files of the latest release. Next, we searched within each file (also in the archives) for the presence of SBOM files, using the constituent parts of SPDX and CycloneDX standards (*e.g.,* `xmlns="http://cyclonedx.org"`, `SPDXID: SPDXRef-DOCUMENT`) as search criterion.

For each project, we then considered the tools used to generate its SBOM and checked the standard representation they adopt. Then, we counted how many times each standard (SPDX or CycloneDX) was used and by which project, making a distinction based on whether the software was related to the domain of SBOMs or not. This was accomplished by looking for the following terms in the repository's name, description, and README file: `SBOM`, `CycloneDX`, `SPDX`, `Bill-of-Materials`, and `Supply-Chain`. This distinction helped us figure out how many of the projects that used SBOMs were directly related to the SBOM domain, and, therefore, not just using SBOM generation tools, but, rather, contributing to the SPDX or CycloneDX ecosystem.

To deepen the study of RQ1, we examined the repository owners' provenance for those who had specified it in their GitHub profile. Since provenance can be specified at different levels and in different ways (*e.g.,* the words `New York City`, `New York`, and `NYC` refer to the same city), we performed a manual pre-processing to standardize provenance at the level of the countries (*e.g.,* United States). Finally, we counted provenance countries with respect to distinct owners and owners' repositories.

2) RQ2: What is the adoption trend of SBOMs by open-source software projects and their owners?: After cloning each repository locally, we collected information about the commit introducing the SBOM generation tool dependency by performing a *git log* operation on the package manager files (*e.g.,* `pom.xml` for Maven, or `requirements.txt` for PyPi) and linearly examining their commit history, from the earliest to the oldest version, until we identified the target commit. Then, for each repository, we considered the date of the commit that introduced the first dependency toward an SBOM generation tool.

We counted the number of adoptions of SBOM generation tools over time with respect to repositories, but also to owners, to control the influence that a few prolific owners might have compared to the whole sample size. We also took into account historically significant events related to SBOMs (*e.g.,* publication of standards or guidelines) to see whether they had an impact on the spread of SBOM generation tools. We categorized each repository on the basis of the year in which it was created, through a 5-year interval scale: *2004-2008*, *2009-2013*, *2014-2018*, and *2019-2023*. According to these categories, we computed some descriptive statistics summarizing when the dependencies toward the SBOM generating tool were added, after how many commits, and how many years, with respect to the intervals representing the creation date of the repository.

To assess how relevant was the adoption of SBOM generation tools, we analyzed the commit message related to the introduction of a dependency on the SBOM tool. The purpose of this analysis was to determine whether any mention of SBOM or SBOM standards was present. To analyze the commit messages, we performed the following steps:

1) We searched into each commit message for instances of the terms `SBOM`, `CycloneDX`, `SPDX`, `Bill-of-Materials`, and `Supply-Chain`
2) Each matched message was deemed relevant to the adoption of SBOMs, the others were manually examined.

3) RQ3: How do open-source projects release SBOM files?: As for RQ2, for each dependency towards an SBOM generation tool, we looked for the presence of SBOM files in the repository and release files by searching for the constituent parts of SPDX and CycloneDX standards. For each SBOM file, we manually verified and included those SBOM files that were actually representative of an SBOM of the project. Finally, we checked for each project whether the SBOM file appeared in the repository, in the release versions, on both sides, or nowhere.

4) RQ4: How frequently are SBOM files updated within open-source software projects?: We analyzed the SBOM files in the repository and the release versions of each software project and, for each of them, we collected:

- The commits that modified each SBOM file present in the repository (and under versioning);
- The releases that published at least an SBOM file.

Then, we analyzed those data from three perspectives:

- For each repository, we counted the commits that modified the SBOM files, starting from the commit in which they were created to the commit on which the analysis of this RQ was performed (*i.e.,* April 1, 2023)—the number of commits in this time interval was then converted in the time interval of a month to estimate SBOM file changes per month;
- For each repository that published SBOMs in the latest release (using the explicit GitHub feature that allows for publishing a release), we computed the percentage of releases containing SBOMs (either in the release archive or as a separate file) out of the total;
- For each commit in which an SBOM file was modified, we checked whether it was associated with a *git tag* and whether that *git tag* was directly associated with a release of the repository—this was to understand whether the generation of a new SBOM file occurred in correspondence with a new release.

TABLE I
LIST OF CONSIDERED SBOM GENERATION TOOLS

Standard	ToolClassification	ToolName	Language	#Dependents
CycloneDX	Build-integration	cyclonedx-cocoapods	Swift, Objective-C	0
		cyclonedx-conan	C, C++	0
		cyclonedx-dotnet	.NET	0
		cyclonedx-gomod	Go	2
		cyclonedx-gradle-plugin	Java	0
		cyclonedx-maven-plugin	Java	74
		cyclonedx-node-module	JavaScript	7
		cyclonedx-node-npm	JavaScript	3
		cyclonedx-php-composer	PHP	5
		cyclonedx-python	Python	64
		cyclonedx-ruby-gem	Ruby	0
		cyclonedx-rust-cargo	Rust	0
		cyclonedx-webpack-plugin	JavaScript	3
		All		*158*
	GitHubAction	cdxgen-action	SourceImages, ContainerImages	0
		gh-cocoapods-generate-sbom	Swift, ObjectiveC	0
		gh-dotnet-generate-sbom	.NET	0
		gh-gomod-generate-sbom	Go	13
		gh-node-module-generatebom	JavaScript	2
		gh-php-composer-generate-sbom	PHP	1
		gh-python-generate-sbom	Python	1
		All		*17*
	Library	cyclonedx-core-java	Java	2
		cyclonedx-dotnet-library	.NET	0
		cyclonedx-go	Go	4
		cyclonedx-javascript-library	JavaScript	0
		cyclonedx-node-pnpm	JavaScript	0
		cyclonedx-node-yarn	JavaScript	0
		cyclonedx-php-library	PHP	0
		cyclonedx-python-lib	Python	1
		All		*7*
SPDX	Build-integration	spdx-maven-plugin	Java	8
		All		*8*
	Library	Spdx-Java-Library	Java	1
		spdx-tools-js	JavaScript	0
		tools	Java	1
		tools-golang	Go	4
		tools-java	Java	0
		tools-python	Python	1
		All		*7*

IV. RESULTS

In this section, we report the results by RQ. A replication package—it contains raw data, scripts, and details on the studied software projects—is available on the web [17]

A. RQ1: Are SBOMs adopted by open-source software projects?

Table I reports the list of all the selected SBOM generation tools from which we retrieved the dependent repositories. Out of 35 SBOM generation tools, only 19 were found to have *dependent* repositories matching the criteria we outlined in Section III-C: the *#Dependents* column indicates the respective count for each SBOM generation tool. The *ToolClassification* column indicates the way in which a given SBOM generation tool is used, namely:

- *Build-integration*, within the build system or within the package manager;
- *GitHub Action*, as part of a CI/CD workflow; or
- *Library*, as a modular component in the source code.

The majority of the *dependents* were associated with build-integration SBOM generation tools (*i.e.,* 166 occurrences in total, of which 8 for SPDX and 158 for CycloneDX), followed by those classified as GitHub Action (*i.e.,* 17 occurrences, all for CycloneDX). It is worth mentioning that 11 repositories depend on more than one SBOM generation tool. Therefore, the total number of distinct repositories that rely on an SBOM generation tool is 186.

Table II reports some descriptive statistics—*i.e.,* mean, Standard Deviation (SD), minimum (min), median, and maximum (max)—about the dependent repositories we studied. The values of these descriptive statistics suggest that the sample is quite heterogeneous in terms of the number of commits, stars, subscribers, contributors, and forks. Furthermore, the number of commits and contributors is, respectively, greater than 579.5 and 4.5 in 50% of the cases (see the median values), so indicating a noticeable collaborative development activity

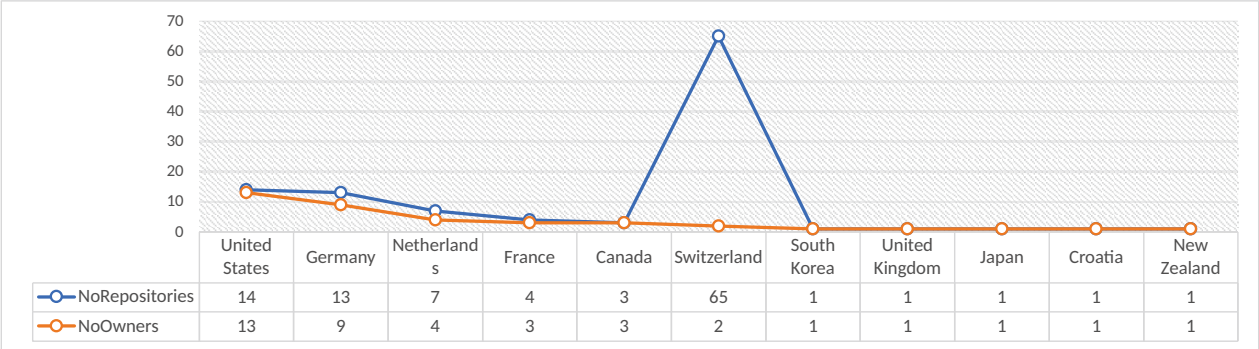

Fig. 1. SBOM generation tools adoptions by owners' provenance

TABLE II
SOME DESCRIPTIVE STATISTICS ABOUT THE PROJECTS DEPENDENT ON SBOM TOOLS

Metric	Mean	SD	Min	Median	Max
NoCommits	2,463.72	7,367.49	104	579.5	63,328
NoStars	937.14	3,674.42	0	3	35,126
NoSubscribers	45.41	186.28	0	3	2,056
NoContributors	35.85	81	1	4.5	364
NoForks	431.7	2,273.05	0	1	26,661

TABLE III
ADOPTION OF SPDX AND CYCLONEDX STANDARDS AMONG THE STUDIED PROJECTS

Domain	Standard	#Repositories
SBOM-related	Only SPDX	8
	Only CycloneDX	22
	Both SPDX and CycloneDX	1
SBOM-unrelated	Only SPDX	3
	Only CycloneDX	151
	Both SPDX and CycloneDX	1
Overall	Only SPDX	11
	Only CycloneDX	173
	Both SPDX and CycloneDX	2

in these software projects.

In Table III, we report information about the adoption of the SBOM standards in the studied software projects. CycloneDX is the most widely used standard, accounting for 94% of the cases (*i.e.,* $(173+2)/186*100$). Two software projects adopted both the CycloneDX and SPDX standards. Also, 31 (17%) software projects using SBOM generation tools are themselves related to the SBOM domain.

Figure 1 depicts the geographical distribution of the owners of open-source software systems that have adopted SBOM generation tools. The United States has the most significant number of distinct owners, followed by two countries of the European Union, namely Germany and the Netherlands (orange line of Figure 1). We can postulate that the high prevalence observed in the cases of the United States (13) and European Union ($17 = 9 + 4 + 3 + 1$) is due to their specific legislation (*i.e.,* President Biden's executive order [10] and the European Union guidelines for securing the Internet of Things [5]). Finally, we can notice a peak in the number of repositories from Switzerland (blue line of Figure 1). This peak is due to a single owner who decided to adopt SBOMs in 61 repositories.

RQ1 summary: SBOMs are not largely adopted in open-source projects and the most used standard is CycloneDX. In the United States and some countries of the European Union (*i.e.,* Germany and the Netherlands), where SBOM is enforced or recommended by government organizations for certain types of software, there is the largest number of distinct owners.

B. RQ2: What is the adoption trend of SBOMs by open-source software projects and their owners?

Figure 2 depicts when, over the evolution histories, repositories and owners first adopted SBOM generation tools. The orange line highlights, month by month, how many new owners first adopted SBOM generation tools for their repositories, while the blue line counts the number of repositories that have started adopting SBOM generation tools. The gray bars indicate major events involving SBOMs and software supply chain practices (*e.g.,* the conception and release of the SPDX and CycloneDX standards, President Biden's executive order, *etc.*).

Based on the data available, SBOM generation tools have started to be adopted by open-source software projects hosted on GitHub since 2014. As shown in Table IV, initially these tools were mainly adopted by those owners who were developing them, *i.e.,* SPDX and CycloneDX (see Table IV).

Since 2021, the interest in SBOM generation tools has increased, coinciding with *(i)* President Biden's executive order [10] and *(ii)* the publication of the SPDX standard as ISO/IEC 5962:2021 [9]. Some owners have been particularly prolific in the adoption of SBOMs for their repositories (blue line of Figure 2), however, it is crucial to note that, overall, an increasing number of new authors are utilizing SBOM generation tools (orange line of Figure 2).

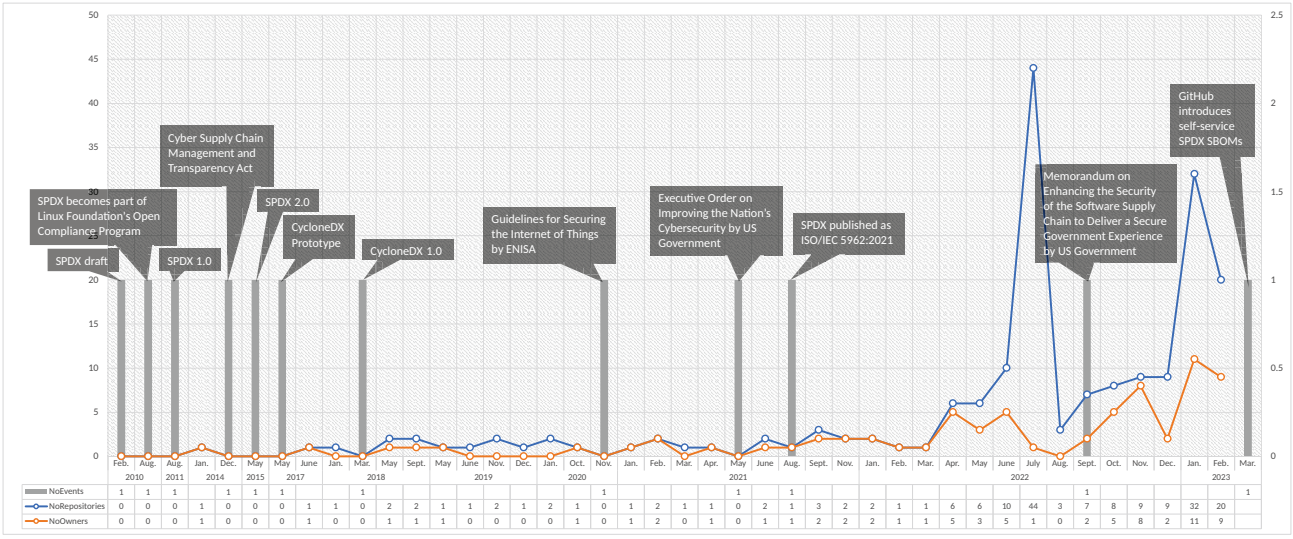

Fig. 2. SBOM generation tools adoption over time by the studied projects

TABLE IV
PROJECTS THAT HAVE ADOPTED SBOM GENERATION TOOLS TILL 2020

Repository	SBOMAdoptionDate
https://github.com/spdx/spdx-maven-plugin	2014-01-13
https://github.com/CycloneDX/cyclonedx-maven-plugin	2017-06-24
https://github.com/spdx/LicenseListPublisher	2018-01-29
https://github.com/stevespringett/Alpine	2018-05-02
https://github.com/CycloneDX/cyclonedx-core-java	2018-05-30
https://github.com/DependencyTrack/dependency-track	2018-09-18
https://github.com/DependencyTrack/hyades-apiserver	2018-09-18
https://github.com/ottlinger/mailclena	2019-05-02
https://github.com/spdx/Spdx-Java-Library	2019-06-09
https://github.com/DependencyTrack/frontend	2019-11-01
https://github.com/CycloneDX/cyclonedx-php-composer	2019-11-19
https://github.com/spdx/spdx-java-rdf-store	2019-12-12
https://github.com/spdx/tools-java	2020-01-06
https://github.com/spdx/spdx-java-jackson-store	2020-01-18
https://github.com/MO-Movia/licit	2020-10-21

Table V reports some descriptive statistics regarding the adoption of SBOM generation tools by software projects, grouping them by the year in which they were created. On average, repositories that were created between 2014 and 2018 are those that first adopted SBOM generation tools, followed by those created between 2019 and 2023, those between 2009 and 2013, and, lastly, those between 2004 and 2008. The number of years and commits from the creation to the adoption of SBOM generation tools are greater for older projects than for more recent ones.

Finally, SBOMs were explicitly mentioned in the commit message that introduced the dependency toward the SBOM generation tool for 90 out of 186 repositories (*i.e.,* in 48% of the studied software projects).

RQ2 summary: Since 2021, there has been an increasing interest in SBOM. Although we do not have causality evidence, this coincides with both President Biden's Executive Order and the publication of the SPDX standard as ISO/IEC 5962:2021.

C. RQ3: How do open-source projects release SBOM files?

Out of 186 software projects, 55 keep the SBOM files under version control only, 14 distribute the SBOM files in release versions only, and 16 use both strategies to distribute SBOM files (*i.e.,* we found the SBOM files both under version control and in release versions). Summing up, SBOM files are available in the repository or published release versions in 46% of the software projects analyzed. For the remaining 101 software projects, we found no SBOM file under versioning, or in release versions. This suggests that either SBOM files are distributed differently or those interested in SBOMs have to generate the SBOM files on their own (*e.g.,* by running the software build).

As for the software projects using GitHub actions to generate SBOM files, we observed that the generation of SBOM files is triggered by a push and pull request in 88% and 59% of the projects, respectively—note that the SBOM file generation workflow can be initiated by different triggers for the same project. Only one project generates an SBOM during the release activity. In other words, in the projects that generate SBOM files by relying on GitHub actions, SBOMs tend to be continuously generated, not only when releasing the software.

RQ3 summary: In 101 out of 186 projects using SBOM tools, SBOM files are not versioned nor attached to release versions, hence distributed differently. In the other cases, it seems that project owners prefer to keep SBOM files under version control rather than including these files in release versions. Finally, projects that leverage GitHub actions to generate SBOM files, in most cases, do so at every push or pull request.

TABLE V
SOME DESCRIPTIVE STATISTICS ON WHEN SBOM GENERATION TOOLS WERE ADOPTED

Statistic	RepositoriesCreationDate	SBOMAdoptionDate	#CommitsToSbomAdoption	#YearsToSbomAdoption
Min	2004-2008	2022-04-13	907	14.25
	2009-2013	2021-04-08	304	9.64
	2014-2018	2014-01-13	1	0
	2019-2023	2019-06-09	1	0
First Quartile	2004-2008	2023-01-11	2,308.75	14.43
	2009-2013	2022-05-25	506.75	10.33
	2014-2018	2021-06-21	195	3.72
	2019-2023	2022-07-09	18	0.02
Median	2004-2008	2023-01-19	9,613.5	15.24
	2009-2013	2023-01-05	2,787.5	11.58
	2014-2018	2022-07-09	573	5.12
	2019-2023	2022-07-22	116	0.76
Third Quartile	2004-2008	2023-02-02	25,411.75	15.86
	2009-2013	2023-01-09	7,985.25	12.84
	2014-2018	2023-01-10	1,475	6.08
	2019-2023	2022-12-01	246	1.87
Max	2004-2008	2023-02-08	62,471	16.47
	2009-2013	2023-02-21	35,326	14.05
	2014-2018	2023-02-28	14,567	8.41
	2019-2023	2023-02-28	5,278	3.96

D. RQ4: How frequently are SBOM files updated within open-source software projects?

Considering those software projects that have SBOM files under version control in their repository, we observed that the number of changes per month is 1.54 on average and 0.96 on median; however, this number is highly variable, as indicated by the SD (1.74), min (0.03), and max (9.72) values. This, overall, suggests a periodic *refresh* of the SBOM, which could be possibly associated with release versions.

Limiting the attention to software projects that enclose SBOM in release versions, SBOMs appear in 61% of the releases of a given software project on average (and 51% on median). We also observed projects that have always put SBOM files in the release versions since they were created.

Finally, we observed that only 2 of all SBOM-publishing release versions are tagged with a commit that modified SBOM files in the repository. There are 8 more tags representing commits in which SBOM files were modified, but no release version is associated with them. Due to the low number of matches, we can infer that there is no correlation between commits modifying SBOM files and release versions.

RQ4 summary: When SBOM files are versioned (or attached to releases) they are in median updated every month or at every release, suggesting, differently from what we observed from the generation of SBOMs by CI/CD, a release-level update.

V. DISCUSSION

In this section, we first discuss the main findings or our study, and then the threats that might affect its validity.

A. Main Findings

Finding #1: A limited number of open-source software projects adopt SBOM generation tools; however, it is progressively increasing, especially among owners from the United States and the European Union.

Although we based our exploratory study on 35 SBOM generation tools officially released by SPDX or CycloneDX—the two major SBOM standards [33]—, we could identify only 186 open-source projects (satisfying the criteria listed in Section III-B) adopting SBOMs. This suggest that SBOMs are still neglected by the open-source community, probably due to the lack of interest and awareness in the software supply chain and its security among software creators, consumers, and end-users. The limited adoption of SBOMs we observed in the open-source community is consistent with what was reported by Xia *et al.* [33]. However, the adoption of SBOMs is growing due to the demand of major players, first and foremost the United States President Biden's executive order. Software creators cannot wait any longer and should begin to identify effective strategies for distributing the SBOMs of their projects. Therefore, we expect that more and more open-source software projects will adopt SBOMs. Future research insights could continue to investigate and monitor this trend of SBOM adoption.

Finding #2: As one of the main barriers to the adoption of SBOM is the availability of tools, different organizations, including standard creators (SPDX and CycloneDX) and forges (GitHub), are making available convenient tools for SBOM creation.

As pointed out by previous literature [33], one of the chal-

lenges restricting the adoption of SBOMs is the limited availability of suitable tools and the features offered by such tools. However, standard creators (SPDX and CycloneDX) and forges (GitHub) are working to make available convenient tools for SBOM generation. In particular, we observed a wider adoption of SBOMs via CI/CD tools (*e.g.,* GitHub action) and build-automation tools (*e.g.,* Maven) as compared to the adoption of SBOMs via libraries. This is probably due to the ease of use of both CI/CD and build-automation tools: the former allows SBOMs to be generated when a certain event occurs (*e.g.,* push) while the latter when building a software artifact. Moreover, we expect an increase in the adoption trend of SBOMs since, recently (*i.e.,* on March 28, 2023, after our query date), GitHub has introduced a new feature to export SBOMs that allows anyone with read access to a GitHub repository to produce an SPDX SBOM by clicking on a button on the dependency graph page of the repository [28]. This self-service feature not only enables on-demand SBOM generation but could also be integrated into CI/CD pipelines and build-automation processes.

Finding #3: Despite it is recommended to *(i)* bundle SBOMs with every release version and that the software supplier has to be responsible for archiving SBOMs, and that *(ii)* SBOMs should be updated at each code change; only a few software projects follow these two recommendations.

According to *NTIA*[1] [18], SBOMs should be bundled with every release version and archived by the software supplier, and be updated at each code change. Furthermore, there is a wide consensus on this matter among the respondents of the Linux Foundation's SBOM readiness survey [14]. Nevertheless, it appears that most of the studied projects ignore or are unaware of such recommendations, thus limiting the potential benefits due to the adoption of SBOMs.

B. Threats To Validity

Below, we discuss the threats that might affect our results with respect to construct, conclusion, and external validity.

Construct Validity. Threats to construct validity concern the relation between theory and observation [32]. The approach we used to detect the adoption of SBOMs might affect the results. In this respect, we could detect the adoption of SBOMs by using one of the following approaches: *(i)* by identifying the use of SBOM generation tools through the GitHub Dependency Graph or *(ii)* by searching for SBOM files within projects hosted on GitHub. We opted for the first approach because a project could use a SBOM generation tool without versioning any SBOM file or attaching SBOM files to any release version (as we observed in 101 out of 186 projects, see Section IV-C). However, the accuracy of the GitHub Dependency Graph might affect the obtained results. To deal with this issue, we ensured that the repositories were indeed *dependent* with respect to the SBOM generation tools by verifying that their dependencies actually appear in their dependency graph and package manager files. This resulted in the exclusion of 11 repositories. Another threat is related to the fact that the GitHub Dependency Graph does not support all build-automation tools (*e.g., Gradle* is not supported by the GitHub Dependency Graph but supported by CycloneDX) and programming languages (*i.e.,* those without a package manager like C, which is not supported by the GitHub Dependency Graph but supported by CycloneDX). The metrics used to answer our RQs might pose another threat to validity. However, there is no accepted metric to quantitatively assess the adoption of SBOMs.

Conclusion Validity. Threats to this kind of validity concern affect the ability to draw the correct conclusions [32]. Our study is purely exploratory, and, therefore, we address our RQs using descriptive statistics and graphical representations.

External Validity. These threats deal with the generalizability of results [32]. The most important threat is that this study is limited to: *(i)* open-source projects hosted on GitHub and *(ii)* SBOM generation tools that support the SPDX and CycloneDX standards.

VI. Conclusion

In this paper, we presented the results of an exploratory mining study that aims to investigate the adoption of Software Bill of Materials (SBOMs) by open-source software projects hosted on GitHub. We mined GitHub to identify repositories that adopt SBOM generation tools developed by SPDX [29] and CycloneDX [3], [22]. We observed that a total of 186 public repositories adopted SBOMs. For these repositories, we collected repository and owner information and analyzed the evolutionary history. The most important takeaway result of our study is that the adoption of SBOMs by open-source projects is still low even if there is an increasing trend. This could be due to the growing interest and pressure from major players, such as the United States Government. We also found that SBOM files are available in the repository or published release versions in 46% of the software projects analyzed, despite it is recommended that SBOMs should be bundled with every release version (and archived by the software supplier), and be updated at each code change.

Based on our initial empirical evidence, we can conclude that SBOMs are getting an increasing attention from software creators and consumers, yet researchers should investigate more thoroughly the root causes behind the limited adoption of SBOM by open-source projects—*e.g.,* by mining information from Q&A forums like *Stack Overflow*. Lastly, there is a pressing need for organizations to update their software to meet the new standards required for the software supply chain. We strongly advocate the provision of convenient SBOM generation tools that can be easily integrated into CI/CD pipelines and build-automation tools. This would enable practitioners to bundle SBOMs within each new release version and ensure that they remain up-to-date with each modification made to the source code.

[1]It is an Executive Branch agency of the Government of the United States.

REFERENCES

[1] G. Bavota, G. Canfora, M. Di Penta, R. Oliveto, and S. Panichella, "How the apache community upgrades dependencies: an evolutionary study," *Empirical Software Engineering*, vol. 20, pp. 1275–1317, 2015.

[2] C. Bogart, C. Kästner, and J. Herbsleb, "When it breaks, it breaks: How ecosystem developers reason about the stability of dependencies," in *Proceedings of IEEE/ACM International Conference on Automated Software Engineering Workshop*. IEEE, 2015, pp. 86–89.

[3] CycloneDX, "CycloneDX Tools," http://archive.md/2023.04.07-162331/https://cyclonedx.org/tool-center/, 2023.

[4] W. Enck and L. Williams, "Top five challenges in software supply chain security: Observations from 30 industry and government organizations," *IEEE Security & Privacy*, vol. 20, no. 2, pp. 96–100, 2022.

[5] European Union Agency for Cybersecurity, "Guidelines for Securing the Internet of Things - ENISA," http://archive.md/2023.04.18-071548/https://www.enisa.europa.eu/publications/guidelines-for-securing-the-internet-of-things/, 2020.

[6] eWeek, "Heartbleed SSL Flaw's True Cost Will Take Time to Tally," http://archive.md/2020.02.10-104151/https://www.eweek.com/security/heartbleed-ssl-flaw-s-true-cost-will-take-time-to-tally, 2020.

[7] GitHub, "About the dependency graph," http://archive.md/2023.04.27-145929/https://docs.github.com/en/code-security/supply-chain-security/understanding-your-software-supply-chain/about-the-dependency-graph, 2023.

[8] H. Hartmann and T. Trew, "Using feature diagrams with context variability to model multiple product lines for software supply chains," in *Proceedings of International Software Product Line Conference*. IEEE, 2008, pp. 12–21.

[9] ISO/IEC, "ISO/IEC 5962:2021 - Information technology — SPDX® Specification V2.2.1," http://archive.md/2023.04.15-133304/https://www.iso.org/standard/81870.html, 2021.

[10] Joe Biden, "Executive Order on Improving the Nation's Cybersecurity," http://archive.md/2022.11.21-161320/https://www.whitehouse.gov/briefing-room/presidential-actions/2021/05/12/executive-order-on-improving-the-nations-cybersecurity/, 2021.

[11] E. Kalliamvakou, G. Gousios, K. Blincoe, L. Singer, D. German, and D. Damian, "The promises and perils of mining github," in *Proceedings of Mining Software Repositories*. ACM, 2014, pp. 92–101.

[12] C. Lamb and S. Zacchiroli, "Reproducible builds: Increasing the integrity of software supply chains," *IEEE Software*, vol. 39, no. 2, pp. 62–70, 2022.

[13] E. Levy, "Poisoning the software supply chain," *IEEE Security & Privacy*, vol. 1, no. 3, pp. 70–73, 2003.

[14] Linux Foundation, "The State of Software Bill of Materials (SBOM) and Cybersecurity Readiness," http://archive.md/2023.04.27-153014/https://www.linuxfoundation.org/research/the-state-of-software-bill-of-materials-sbom-and-cybersecurity-readiness, 2022.

[15] R. A. Martin, "Visibility & control: addressing supply chain challenges to trustworthy software-enabled things," in *Proceedings of IEEE Systems Security Symposium*. IEEE, 2020, pp. 1–4.

[16] S. Nadgowda, "Engram: the one security platform for modern software supply chain risks," in *Proceedings of International Workshop on Container Technologies and Container Clouds*. ACM, 2022, pp. 7–12.

[17] S. Nocera, S. Romano, M. Di Penta, R. Francese, and G. Scanniello, "Software Bills of Material Adoption: A Mining Study from GitHub - A Replication Package," https://figshare.com/s/50b3f91580e8f4120189, 2023, doi: https://doi.org/10.6084/m9.figshare.22626490.v1.

[18] NTIA Multistakeholder Process on Software Component Transparency Standards and Formats Working Group, "SBOM Options and Decision Points," http://archive.md/2023.04.27-151750/https://www.ntia.gov/files/ntia/publications/sbom_options_and_decision_points_20210427-1.pdf, 2021.

[19] ——, "Survey of Existing SBOM Formats and Standards," http://archive.md/2023.04.27-151457/https://www.ntia.gov/files/ntia/publications/sbom_formats_survey-version-2021.pdf, 2021.

[20] M. Ohm, H. Plate, A. Sykosch, and M. Meier, "Backstabber's knife collection: A review of open source software supply chain attacks," in *Proceedings of International Conference on Detection of Intrusions and Malware & Vulnerability Assessment*. Springer, 2020, pp. 23–43.

[21] Open Web Application Security Project, "A06 Vulnerable and Outdated Components - OWASP Top 10:2021," http://archive.md/2023.04.06-063141/https://owasp.org/Top10/A06_2021-Vulnerable_and_Outdated_Components/, 2021.

[22] OWASP, "CycloneDX," https://cyclonedx.org, 2023.

[23] PyGithub, "PyGithub documentation," http://archive.md/2023.04.11-142811/https://pygithub.readthedocs.io/en/latest/index.html, 2023.

[24] R. Shirey, "Internet security glossary, version 2," RFC Series, Tech. Rep., 2007.

[25] Software Package Data Exchange (SPDX), "SPDX Tools," http://archive.md/2023.04.07-162310/https://spdx.dev/spdx-tools/, 2023.

[26] D. Spadini, M. Aniche, and A. Bacchelli, "Pydriller: Python framework for mining software repositories," in *Proceedings of Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. ACM, 2018, pp. 908–911.

[27] Stack Overflow, "Developer survey," http://archive.md/2022.06.22-155802/https://survey.stackoverflow.co/2022/, 2022.

[28] The GitHub Blog, "Introducing self-service SBOMs," http://archive.md/2023.04.24-070357/https://github.blog/2023-03-28-introducing-self-service-sboms/, 2023.

[29] The Linux Foundation, "The Software Package Data Exchange (SPDX)," https://spdx.dev, 2023.

[30] US Department of Commerce, "The minimum elements for a software bill of materials (SBOM)," http://archive.md/2023.04.18-062538/https://www.ntia.doc.gov/files/ntia/publications/sbom_minimum_elements_report.pdf, 2021.

[31] D. L. Vu, I. Pashchenko, F. Massacci, H. Plate, and A. Sabetta, "Towards using source code repositories to identify software supply chain attacks," in *Proceedings of ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2020, p. 2093–2095.

[32] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslén, *Experimentation in Software Engineering*. Springer, 2012.

[33] 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 *45th IEEE/ACM International Conference on Software Engineering, ICSE 2023, Melbourne, Australia, May 14-20, 2023*. IEEE, 2023, pp. 2630–2642.

[34] N. Zahan, E. Lin, M. Tamanna, W. Enck, and L. Williams, "Software bills of materials are required. are we there yet?" *IEEE Security & Privacy*, vol. 21, no. 2, pp. 82–88, 2023.

[35] N. Zahan, T. Zimmermann, P. Godefroid, B. Murphy, C. Maddila, and L. Williams, "What are weak links in the npm supply chain?" in *Proceedings of International Conference on Software Engineering: Software Engineering in Practice*. ACM, 2022, p. 331–340.